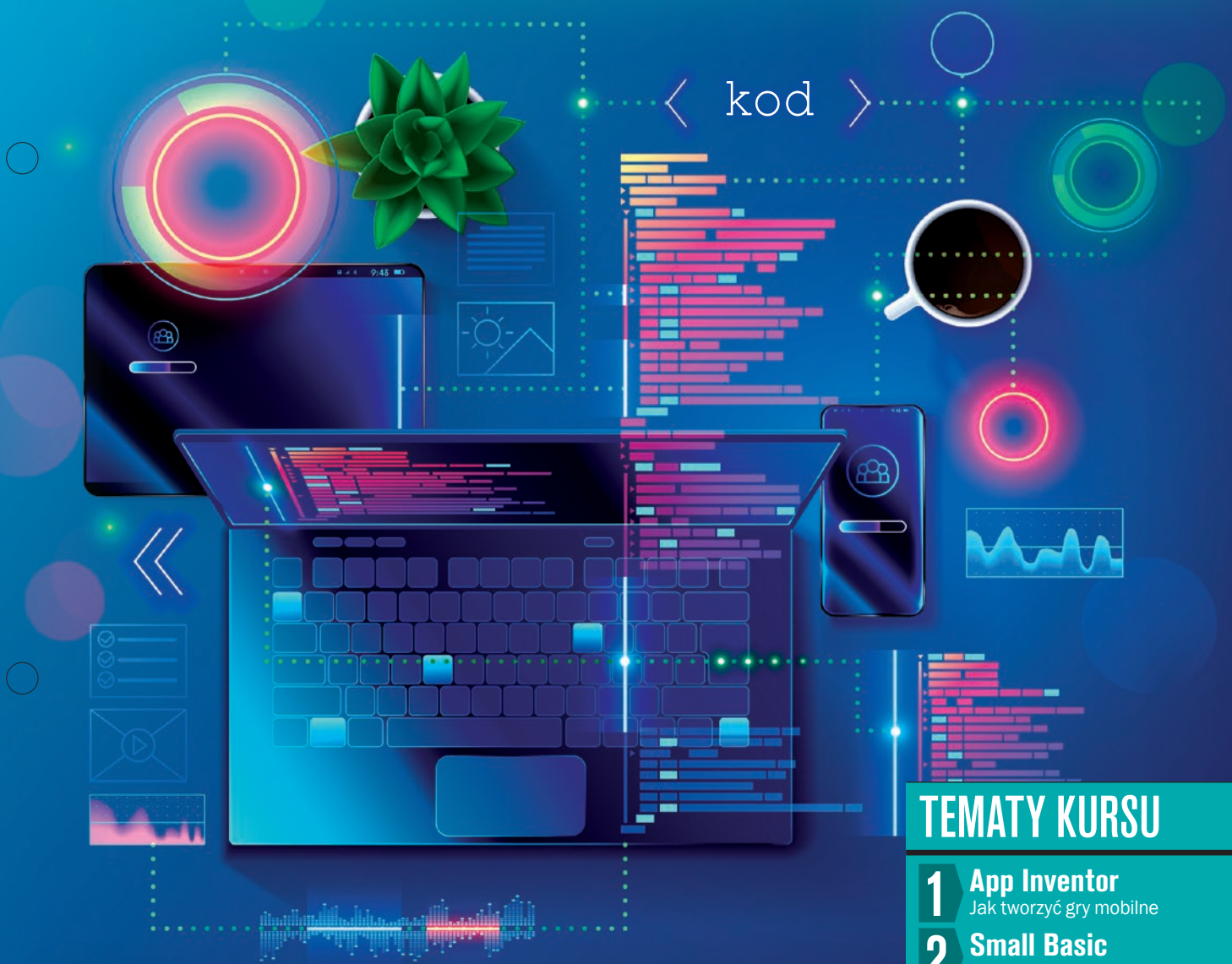
KURS PROGRAMOWANIA

W tym kursie będziemy stopniowo poznawać tajniki programowania. Każdy odcinek to inny projekt, opisany krok po kroku, który wprowadza początkującego programistę w znane środowiska programistyczne



Programowanie – choć jest tylko jednym słowem, kryje się pod nim ogrom znaczeń, a raczej sposobów na to, jak jest ono postrzegane. W tym kursie zobaczymy różne oblicza programowania. Każdy z odcinków będzie przedstawiał inne środowisko programistyczne i język programowania. Po to, by nauczyć się tego, co w programowaniu uniwersalne i wspólne dla różnych środowisk i języków. Zaczniemy od zagadnień prostszych, aby wykonywać kolejne kroki w głąb programistycznego świata. Będziemy płynnie przechodzić od środowisk edukacyjnych do profesjonalnych języków (między innymi C#, Visual Basic) i narzędzi (między innymi Visual Studio, Unity Engine). Środowiska edukacyjne są oczywiście prostsze do opanowania od profesjonalnych – jednak w pełni funkcjonalne. Pozwalają na tworzenie gier i aplikacji dokładnie takich, jakie sobie wymyślimy. **kj**

TEMATY KURSU

1. **App Inventor**
   Jak tworzyć gry mobilne
2. **Small Basic**
   Jak zrobić Tetris
3. **Visual Basic**
   Wstęp do profesjonalnych języków programowania
4. **Visual Basic**
   Platforma .NET
5. **C#**
   Język do wielu zastosowań
6. **Unity**
   Jak zrobić własną grę z grafiką 3D

Fot. Andrey Suslov/istockphoto.com

APP INVENTOR: JAK TWORZYĆ GRY MOBILNE

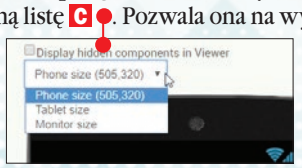
Kurs rozpoczynamy od środowiska **App Inventor**. To oprogramowanie, dostępne z poziomu przeglądarki, pozwala na tworzenie gier i aplikacji przeznaczonych do instalacji na urządzeniach mobilnych z systemem Android (trwają beta testy wersji do aplikacji na iOS).
Jest coś, co wyróżnia App Inventora od pozostałych narzędzi i języków, które zostaną omówione w kolejnych częściach kursu. Programując w App Inventorze, nie musimy pisać skryptów – możemy układać je z puzzli. Czyni to App Inventora idealnym środowiskiem do rozpoczęcia przygody z programowaniem. Dzięki wykorzystaniu puzzli unikniemy literówek w skryptach, które są częstą bolączką początkujących programistów.

OKNO PROGRAMU

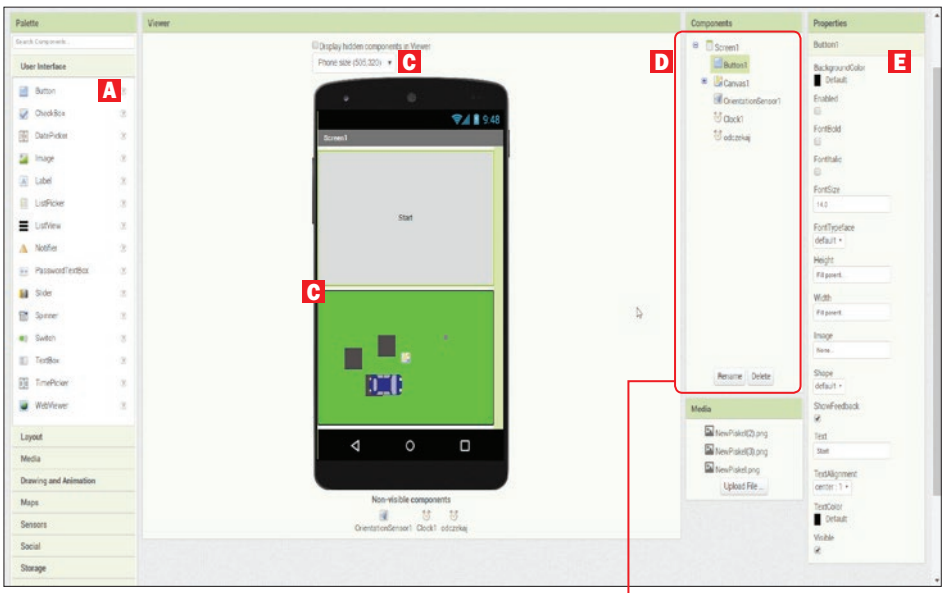
Zanim przejdziemy do tworzenia własnej gry, poznajmy podstawowe pojęcia związane z programem App Inventor – ułatwi to nam dalszą pracę.
Okno programu App Inventor składa się z dwóch głównych ekranów. Pierwszy z nich to **Designer**, gdzie modelujemy wygląd naszej aplikacji, a drugi to **Blocks**, gdzie przypisujemy jej działanie, tworząc skrypty zbudowane z puzzli zwanych tu blokami. Pomiędzy ekranami przełączamy się, klikając na przyciski odpowiadające ich nazwom, widoczne w prawym górnym rogu okna ●.

DESIGNER

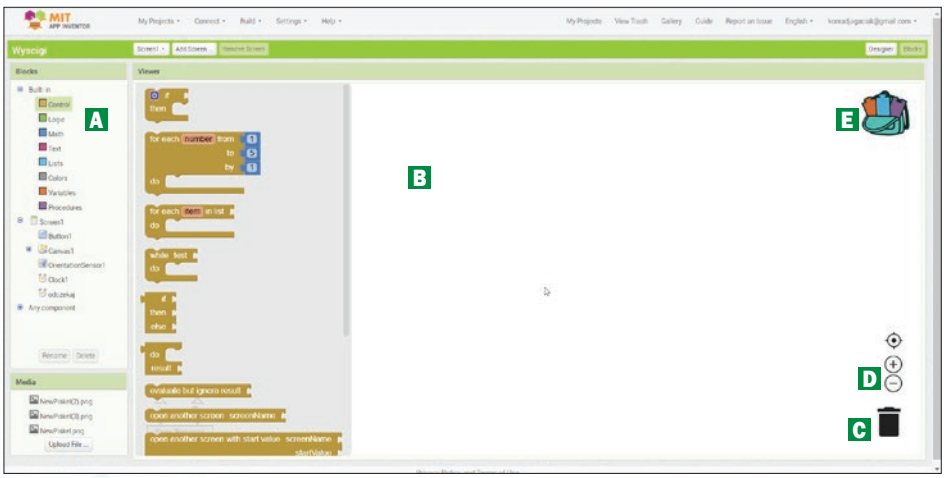
Zajmijmy się najpierw ekranem Designer. Pierwsza od lewej strony sekcja to paleta komponentów **A**. Komponenty to elementy, które umieszczamy w programie – na przykład przyciski, pola tekstowe – ale też moduły pozwalające na wykorzystanie map, użycie dyktafonu, aparatu czy żyroskopu.
Potrzebne nam komponenty przeciągamy na ekran podglądu **B** – on pokazuje nam, jak będzie wyglądała nasza aplikacja, kiedy włączymy ją na urządzeniu z Androidem.
Nieco ponad podglądem widzimy niewielką rozwijaną listę **C** ●. Pozwala ona na wybranie



docelowego urządzenia, na jakim ma być uruchamiana nasza aplikacja. Do wyboru mamy telefon, tablet i monitor – zmiana urządzenia powoduje jedynie zmianę wielkości ekranu podglądu i nie wpływa na efekt naszej pracy. Nawet jeśli pracujemy na podglądzie dla tabletów, stworzone przez nas aplikacje i tak będzie można uruchomić na telefonie.
Kolejna sekcja – po prawej stronie podglądu – to zestawienie dodanych do projektu komponentów **D**. Kiedy dodamy komponent do programu, wyświetli się on na liście ●. Wygląd i działanie komponentu możemy zmieniać poprzez zmianę wartości jego właściwości **E**. Są one wypisane w ostatniej z sekcji, skrajnie po prawej stronie.

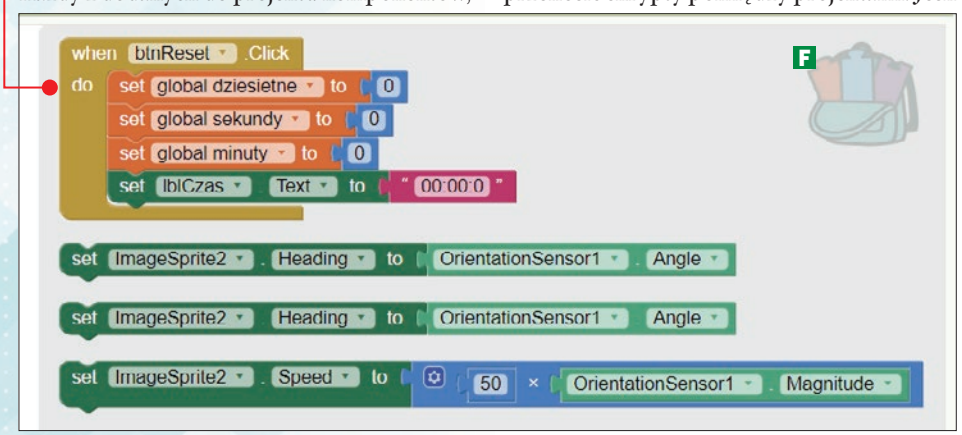


jest wyświetlany w programie tak samo jak standardowe kategorie dostępnych puzzli. W ten właśnie sposób komponenty poszerzają możliwości oprogramowania, dając nam do dyspozycji dodatkowe puzzle. W prawym dolnym rogu widzimy ikonę kosza na śmieci **C**.



BLOCKS

Po przejściu do sekcji Blocks po lewej stronie okna widzimy kategorie **A** dostępnych bloków i duże białe pole **B**. Wybierając kategorię, widzimy dostępne w niej puzzle, które możemy przeciągać na białe pole, tworząc w ten sposób skrypty ● opisujące działanie naszej aplikacji.
Każdy z dodanych do projektu komponentów, jest wyświetlany w programie tak samo jak standardowe kategorie dostępnych puzzli. W ten właśnie sposób komponenty poszerzają możliwości oprogramowania, dając nam do dyspozycji dodatkowe puzzle. W prawym dolnym rogu widzimy ikonę kosza na śmieci **C**.
Przeciągając na nią umieszczone na białym polu skrypty, możemy je usunąć.
Powyżej kosza widoczne są przyciski z plusem i minusem **D**. Służą one do powiększania i pomniejszania puzzli.
Jeszcze nieco wyżej widzimy ikonę plecaka **E**. Plecak to mechanizm, dzięki któremu możemy przenosić skrypty pomiędzy projektami. Jeśli



Wyrwij, użyj dziurkacza i zbieraj do segregatora

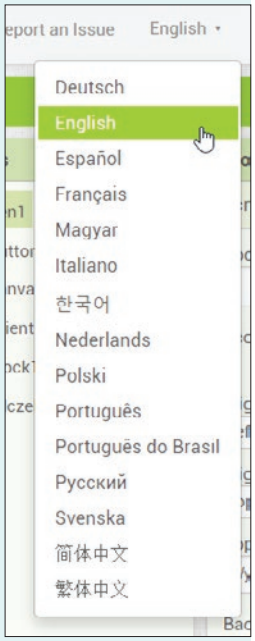
przeciągniemy skrypt do plecaka, zostanie on do niego dodany. Klikając na ikonę plecaka, otwieramy go, czyli wyświetlamy znajdujące się w nim skrypty F. Jeśli chcemy wykorzystać skrypt znajdujący się w plecaku w swoim projekcie, przeciągamy go z plecaka na obszar roboczy. Zawartość plecaka przypisana jest do konta Google, z którego logujemy się do programu.

Przed przystąpieniem do tworzenia gier dobrze jest zapoznać się z dostępnymi puzzlami (blokami), aby łatwiej było nam się poruszać po oprogramowaniu.

Po najechaniu na blok wyświetlona zostanie podpowiedź, opisująca jego działanie.

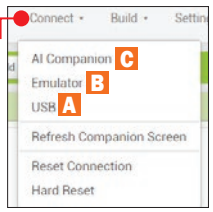
Wersja językowa

App Inventor ma polską wersję językową. Nie jest ona jednak polecana dla osób, które chcą wykorzystać to oprogramowanie jako jeden z pierwszych kroków do dalszej nauki programowania. Zdecydowana większość języków programowania jest oparta na języku angielskim. Bloki, z jakich korzystamy w App Inventorze do budowania skryptów, są odzwierciedleniem programistycznych instrukcji, które spotkamy potem w innych narzędziach, podczas pisania skryptów. Dzięki wykorzystaniu bloków w języku angielskim łatwiej zauważymy analogie pomiędzy blokami a odpowiadającymi im programistycznymi instrukcjami.



JAK TESTOWAĆ GRĘ

Podczas tworzenia gier warto testować poszczególne funkcje wprowadzane przez nas do projektu. Aby móc to zrobić, niezbędne jest nawiązanie połączenia pomiędzy App Inventorem a fizycznym urządzeniem z Androidem bądź emulatorem Androida. Opcje służące do połączenia uruchomimy po wybraniu z menu pozycji **Connect**.



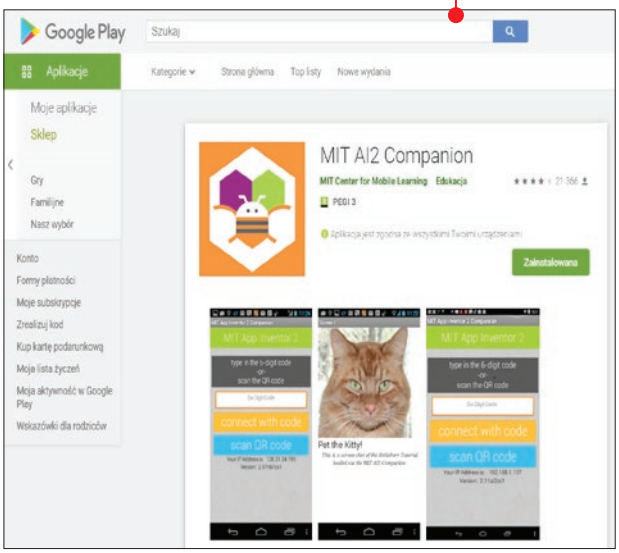
- Opcja **USB** A pozwala na przetestowanie tworzonego przez nas projektu na podłączonym urządzeniu z Androidem, w którym włączono tryb debuggowania.
- Opcja **Emulator** B pozwala na przetestowanie projektu za pomocą wirtualnego urządzenia. Niezbędna jest do tego instalacja odpowiedniego emulatora – **aiStarter**.
- Rekomendowanym przez twórców App Inventora rozwiązaniem jest jednak wykorzystanie opcji **AI Companion** C.

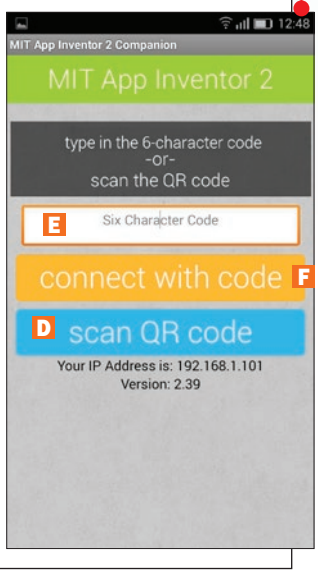
1 Po jej wybraniu wyświetlony zostanie kod QR.

2 Możemy zeskanować go poprzez aplikację **MIT AI2 Companion** (do pobrania ze Sklepu Play).

3 Skaner uruchamiamy za pomocą przycisku **Scan QR code** D. Możemy również ręcznie wpisać tekst wyświetlony obok kodu QR w pole **Six Character Code** E.

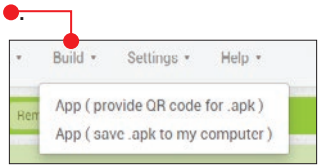
4 Następnie, naciskając **Connect with code** F, łączymy się z naszym projektem. Co ważne – wprowadzane przez nas w projekcie zmiany będą od razu widoczne w testowanej aplikacji.





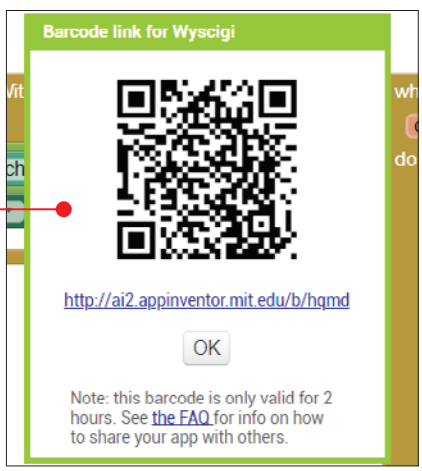
JAK ZAINSTALOWAĆ GOTOWĄ GRĘ NA URZĄDZENIU Z ANDROIDEM

Aby zainstalować aplikację na urządzeniu z systemem Android, niezbędne jest posiadanie pliku **APK**. Do jego wytworzenia możemy wykorzystać opcje, które znajdziemy w menu **Build**.



1 Pierwsza z nich – **App (provide QR code for .apk)** – pozwala na wygenerowanie kodu QR, który podobnie jak w przypadku testowania, możemy zeskanować przez aplikację MIT AI2 Companion. Pozwala to na pobranie pliku APK do naszego urządzenia mobilnego i rozpoczęcie instalacji. W przypadku plików APK możliwe jest skanowanie kodu QR także innymi skanerami – ponieważ kod prowadzi po prostu do pliku z projektem, więc niezależnie od użytego skanera będziemy mogli pobrać ten plik.

Kod QR jest aktywny przez dwie godziny od momentu wygenerowania.



2 Druga z widocznych w menu opcji – **App (save .apk to my computer)** – generuje plik APK i pozwala na zapisanie go na dysku naszego komputera. Taki plik przesyłamy w dowolny sposób do urządzeń mobilnych i otwierając go, uruchamiamy instalację.

WYŚCIGI SAMOCHODOWE NA ANDROIDA

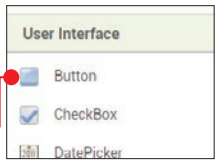
Kiedy wiemy już, do czego służą poszczególne elementy programu, możemy przejść do tworzenia naszej pierwszej gry mobilnej. Będą to **Wyścigi samochodowe**. Zadaniem gracza będzie dojechanie do mety w jak najkrótszym czasie, uważając na to, by nie wyjechać poza trasę. Samochód gracza będzie sterowany poprzez odpowiednie nachylenie urządzenia.

App Inventor 2 pozwala na tworzenie skryptów w projekcie poprzez interfejs oparty na puzzlach, które podzielone są na kategorie. Niektóre z puzzli są jednak przypisane do poszczególnych komponentów. Wykorzystanie takich puzzli jest możliwe dopiero po przeciągnięciu komponentów do projektu. Dlatego najpierw należy przygotować wygląd okna naszej gry, a dopiero potem przejść do tworzenia skryptów. Tego rodzaju podejście występuje także w wielu profesjonalnych środowiskach programistycznych.

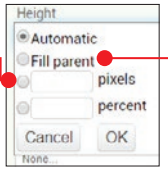
MODELOWANIE OKNA GRY

Zadajmy sobie pytanie: Co ma zobaczyć użytkownik naszej gry, tuż po jej włączeniu? Dobrze, aby był to przycisk, którym można rozpocząć rozgrywkę.

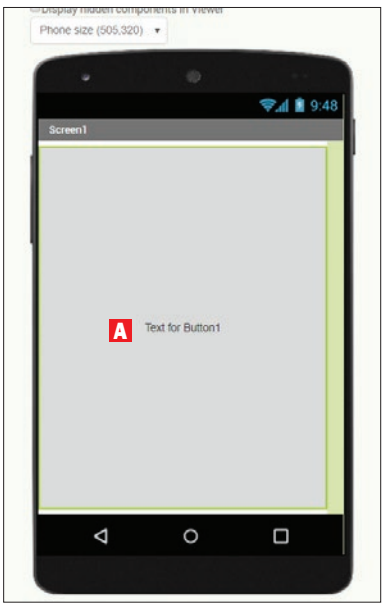
1 By go stworzyć, wykorzystamy tradycyjny przycisk – komponent **Button**. Znajdziemy go w palecie komponentów, w zakładce **User Interface**. Przeciągamy przycisk na podgląd okna programu.



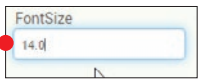
2 Przycisk nie przypomina jeszcze tego, co chcemy uzyskać w naszej grze. Chcielibyśmy, aby przycisk miał inną formę. By był możliwie dużym prostokątem, który wypełnia cały ekran. Po zaznaczeniu odpowiedniego komponentu na oknie podglądu projektu, możemy modyfikować jego właściwości (**Properties**). Tak też musimy zrobić z naszym przyciskiem. Pierwsza zmiana to wymiary przycisku. Pola **Height** i **Width** pozwalają nam na określenie odpowiednio wysokości i szerokości przycisku. Możemy na przykład wpisać tam konkretne wymiary, wyrażone w pikselach. Niestety, nie znamy wymiarów ani rozdzielczości ekranów, na jakich uruchamiana będzie nasza gra. Możemy jednak sprawić, by gra wyglądała dobrze na każdym urządzeniu, na jakim zostanie uruchomiona, by przycisk wypełniał cały ekran, dopasowując swoją wielkość do jego rozmiarów. Zarówno wysokość, jak i szerokość przycisku powinna dostać wartość **Fill parent**. Przycisk wypełnia teraz cały obszar ekranu gry **A**.



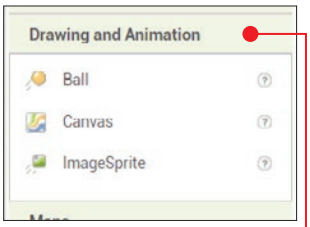
3 Kolejną zmianą, jaką należy wprowadzić, jest usunięcie z przycisku napisu **Text for Button1** i zastąpienie go pasującym do gry słowem – na przykład **Start** lub **Rozpocznij**. W tym celu usuwamy we właściwościach przycisku tekst wpisany w pole **Text**. Zastępujemy go napisem, jaki chcemy widzieć na przycisku.



4 Wyświetlony na przycisku tekst jest dość mały. Możemy powiększyć go, wpisując interesujący nas rozmiar czcionki w pole **FontSize** we właściwościach przycisku.



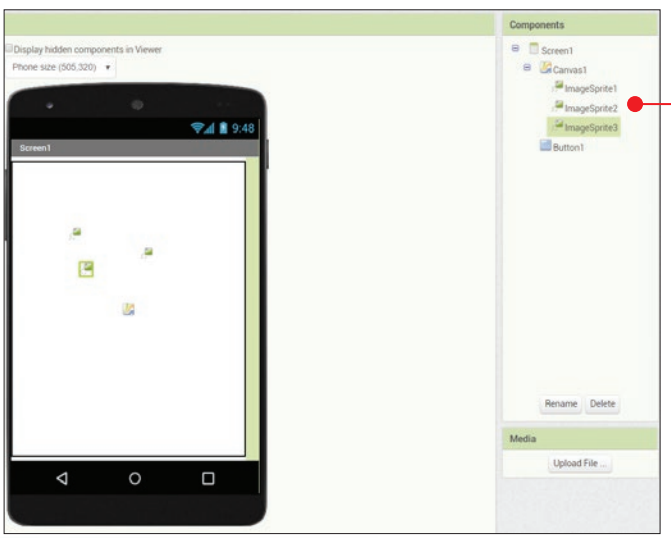
5 W ten sposób przycisk do rozpoczynania naszej gry jest już gotowy. Co dalej powinno znaleźć się na ekranie? Przycisk powinien zniknąć, a powinna go zastąpić trasa, którą musimy przejechać, nasz samochód i meta – innymi słowy, główne elementy graficzne naszej gry. Takie elementy gry nazywamy „duszkami" (ang. sprite). Duszki to komponenty o nazwie **ImageSprite**. Jeśli spróbujemy przeciągnąć komponent znajdujący



się w kategorii **Drawing and Animation** bezpośrednio do okna, w którym modelujemy nasz projekt, nie zostanie on dodany. Komponenty **ImageSprite** można dodawać tylko wewnątrz komponentu **Canvas**. Dlatego to właśnie Canvas należy dodać najpierw do naszego projektu. Zanim dodamy Canvas, trzeba byłoby jeszcze ukryć przycisk. Możemy to zrobić, usuwając w jego właściwościach zaznaczenie przy pozycji **Visible**.

6 Gdy przycisku już nie widać, można dodać Canvas do projektu. Aby gra była widoczna na całym ekranie urządzenia mobilnego, Canvas powinien mieć wymiary takie jak przycisk – wypełniając cały możliwy obszar. Dlatego we właściwościach obiektu **Canvas** zmieniamy jego wysokość i szerokość (odpowiednio Height i Width) na **Fill parent**, jak w przypadku przycisku.

7 Na powiększony do pełnych rozmiarów **Canvas** przeciągamy trzykrotnie komponent **ImageSprite**. Te duszki wykorzystamy do budowy toru, po którym będzie jeździł nasz samochód. Każdy z tych trzech duszków będzie fragmentem toru, a poprzez skrypty ustalimy jego dokładny wygląd. Na

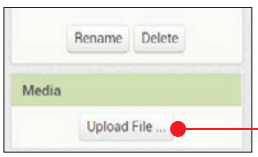


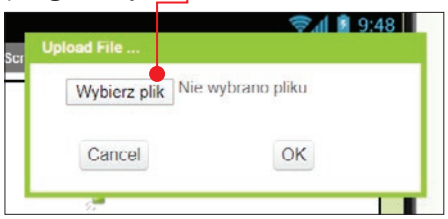
Piskel

Do tworzenia grafik polecamy oprogramowanie Piskel (**W KŚ+**). Jest to edytor graficzny stworzony z myślą o pixel art (sposób tworzenia grafiki rastrowej za pomocą programów pozwalających na edytowanie obrazów na poziomie pojedynczych pikseli).

tym etapie niezbędne będzie jednak wczytanie grafiki do tych obiektów. Najlepiej przygotować plik graficzny jednolitego koloru – takiego, jakiego chcemy mieć trasę.

8 Plik graficzny należy wczytać do App Inventora. Robimy to w niewielkiej sekcji **Media**, która znajduje się poniżej listy dodanych już do projektu komponentów. Należy kliknąć na przycisk **Upload file**, by otworzyć okno importu plików, w którym klikamy na **Wybierz plik**, co otwiera okno dialogowe, z poziomu którego możemy wczytać plik graficzny.

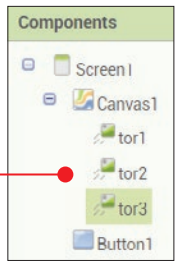


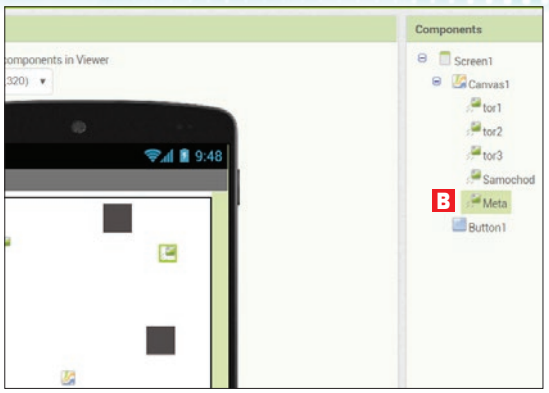


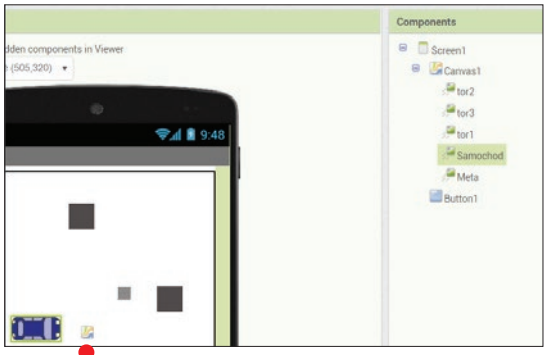
9 Gdy plik jest wczytany i zatwierdzony, możemy przypisać go do duszków. W każdym z nich powinniśmy we właściwości **Picture** wybrać wczytany plik graficzny.

10 Pod listą dodanych komponentów, a nad sekcją **Media**, znajduje się przycisk **Rename**. Jego użycie pozwala na zmianę nazwy zaznaczonego komponentu dodanego do projektu. Zmieniamy nazwy duszków, z których będziemy budować tor, na tor1, tor2 i tor3.

11 Dodajemy dwa kolejne komponenty typu **ImageSprite** i zmieniamy im nazwy na **Samochod** i **Meta** B.

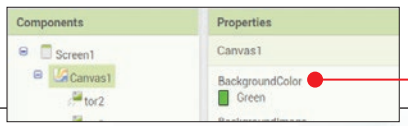






12 Wczytujemy dwa kolejne pliki graficzne – widziany z lotu ptaka samochód (zwrócony w prawą stronę) i metę. Meta może być kwadratem w biało-czarną szachownicę. Wczytanie grafiki przypisujemy (poprzez właściwość **Picture**) do dodanych w poprzednim kroku duszków.

13 Dla obiektu **Canvas** możemy jeszcze zmienić wartość właściwości **BackgroundColor** – mówiącej o kolorze tła. Wybieramy kolor zielony, aby tło symbolizowało trawę wokół drogi.

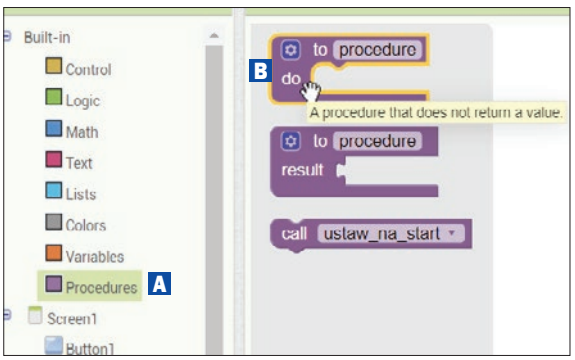


UKŁADANIE SKRYPTÓW

UŁOŻENIE OBIEKTÓW W OKNIE

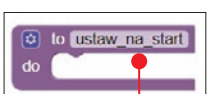
Przechodzimy do ekranu **Blocks** – to w nim należy ułożyć skrypty opisujące działanie gry.

1 Zaczniemy od odpowiedniego modelowania toru. Z trzech kwadratów musimy stworzyć tor do jazdy. Tor powinien mieć kształt odwróconej litery U – z dwoma pionowymi odcinkami po bokach i jednym, poziomym, u góry okna.

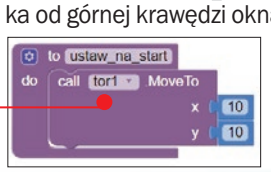


2 Ułożymy procedurę, która zmieni rozmiar obiektów tor1, tor2 i tor3, rozciągając je, i przesunie je w odpowiednie miejsca, tak by utworzyły opisany wcześniej tor. By stworzyć procedurę z kategorii **Procedures** A, bierzemy blok **to procedure do** B i przeciągamy go na pusty obszar roboczy.

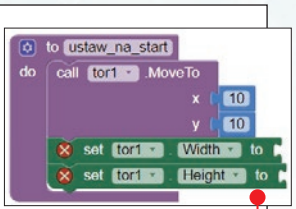
3 W edytowalnym polu tekstowym na bloku wpisujemy nazwę procedury: **ustaw_na_start**.



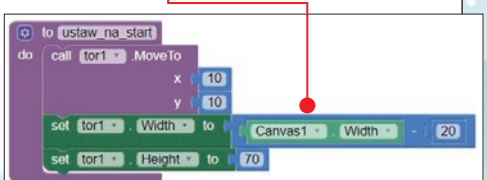
4 Do wnętrza procedury wkładamy blok **call tor1.MoveTo**, a do niego przyłączamy podwójnie (w pola **x** i **y**) blok pozwalający na wpisanie liczby, którą znajdziemy w kategorii **Math** (gdzie domyślnie wpisane jest 0). Pole **x** mówi o odległości duszka od lewej krawędzi okna, pole **y** zaś o odległości duszka od górnej krawędzi okna. W obydwa pola wpisujemy liczbę **10**. Pierwszy element toru będzie poziomą linią u góry okna.



5 Pierwszy fragment toru jest już przesunięty w odpowiednie miejsce, jednak trzeba mu jeszcze nadać odpowiednie wymiary. Zrobimy to, używając **set tor1.Width to** i **set set tor1.Height to**, które służą odpowiednio do ustawiania szerokości oraz wysokości duszka.

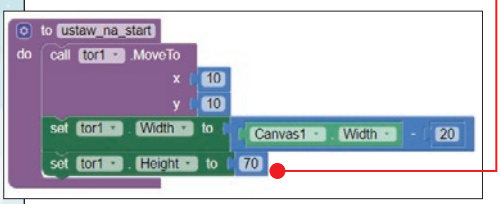


6 Ponieważ ustawiliśmy nasz tor 10 pikseli od lewej krawędzi, aby był on symetryczny, powinien się kończyć 10 pikseli przed prawą krawędzią. A dwa razy po 10 pikseli po bokach daje nam 20 pikseli. Szerokość duszka powinna być 20 pikseli mniejsza niż szerokość obiektu **Canvas**. Możemy to ułożyć za pomocą bloku odejmowania, który znajduje się w kategorii **Math**, a także bloku **Canvas1.Width** (zawierającego szerokość

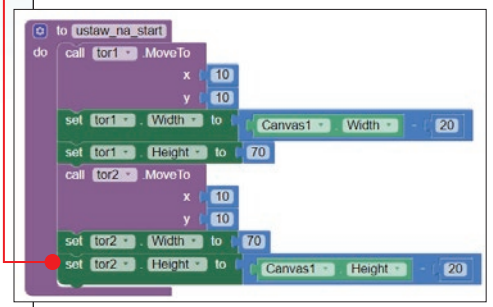


kanwy), znajdującego się w kategorii **Canvas1** i bloku z liczbą do wpisania.

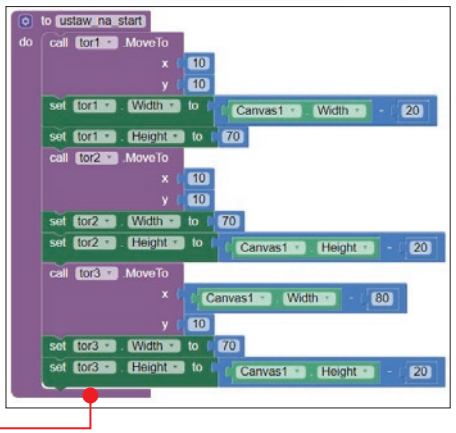
7 Również wysokość obiektu **tor1** powinniśmy ustalić. Tu możemy zdecydować się na wpisanie odpowiadającej nam liczby – **70** pikseli powinno być odpowiednie.



8 Analogicznie do poziomej części toru, w obiekcie **tor2** powinniśmy ustawić lokalizację i wymiary tak, by ten stał się pionową częścią toru po lewej stronie okna gry.

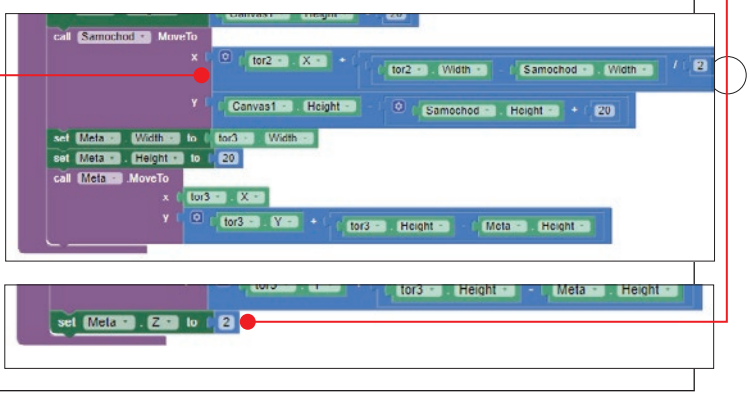


9 Także trzeci fragment toru powinniśmy ustawić po prawej stronie, w takim miejscu, by utworzył on symetryczny obraz do linii po lewej stronie.



10 Również **Samochod** i **Meta** powinny być przeniesione w odpowiednie miejsca. Samochód powinien znaleźć się na dole prostej po lewej stronie. Obliczamy jego **x** i **y**, tak by znalazł się on idealnie na środku, jeśli chodzi o szerokość, i by grafika samochodu kończyła się 10 pikseli przed końcem toru, jeśli chodzi o pionową oś. Obliczenia należy ułożyć z bloków znajdujących się w kategorii **Math**.

11 Meta powinna znaleźć się na drugim końcu toru – niech ma ona 20 pikseli wysokości i szerokość taką jak tor. Powinna być ustawiona w takim miejscu, by jej dolna krawędź pokrywała się z dolną krawędzią toru.



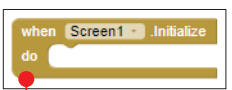
12 Dla mety należy ustawić też właściwość **Z** – poprzez blok **set Meta.Z to** – na wartość **2**. Domyślnie każdy **ImageSprite** ma właściwość **Z** ustawioną na **1**. Im większa jest tam wartość, tym „wyżej" jest obiekt. Jeśli meta ma zasłaniać końcówkę toru, powinna mieć **Z** większe niż on. Inaczej meta będzie przysłonięta torem.

Pamiętaj

Jeśli masz grafikę samochodu w rozmiarze innym niż 60 na 30 pikseli – zmień wymiary obiektu **Samochod** poprzez użycie odpowiednich bloków – tak jak zmieniasz wymiary torów – niech ma on 60 pikseli szerokości i 30 pikseli wysokości.

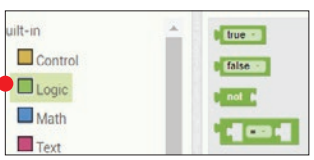
UKRYCIE CANVAS PRZED STARTEM GRY

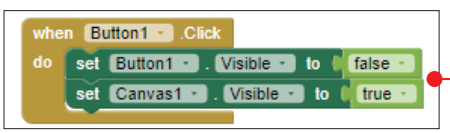
Kolejnym krokiem w tworzeniu naszej gry będzie sprawienie, by na początku, tuż po uruchomieniu, widoczny był jedynie przycisk, a **Canvas** był ukryty.



1 W tym celu korzystamy z bloku **when Screen1.Initialize**, który zgłasza zdarzenie uruchomienia programu. Bloki o tym kolorze, których nie da się od dołu i od góry przyczepić do innych bloków, to zdarzenia. Zdarzenia mówią o wystąpieniu czegoś w programie i pozwalają nam na reakcję na tego czegoś wystąpienie. To co włożymy do bloku **when Screen1.Initialize**, wykona się, gdy uruchomimy grę.

2 Właściwość **Visible** zarówno dla **Button1**, jak i **Canvas1** może przyjąć tylko jedną z dwóch wartości – true lub false. Bloki z tymi wartościami znajdują się w kategorii **Logic**. Pierwszy z nich oznacza prawdę, a drugi fałsz. Blok **false** przyczepiamy do **set Button1.Visible to**, a blok **true** doczepiamy do ustawienia widoczności obiektu **Canvas1**.

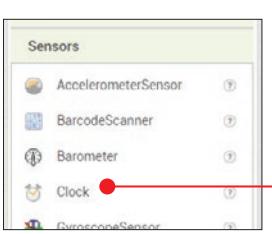




3 Przycisk ma zniknąć, gdy na niego klikniemy, a **Canvas** ma się wtedy pojawić. Z kategorii **Button1** bierzemy blok **when Button1.Click do** i układamy w nim skrypt ukrywający przycisk i pokazujący obiekt **Canvas1**.

DODAWANIE NIEWIDOCZNYCH KOMPONENTÓW

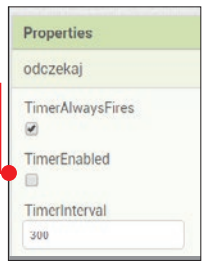
Gdybyśmy zaraz pod blokiem odpowiedzialnym za pokazywanie komponentu **Canvas** uruchomili procedurę **ustaw_na_start**, obiekty mog łyby pojawić się w nieodpowiednich miejscach. To przez opóźnienia programu, jeśli w skrypcie odczytujemy z komponentu **Canvas** wymiary, mogą być one jeszcze niezaktualizowane w momencie wywoływania procedury i mogą zostać źle odczytane. Powinniśmy zatem odczekać chwilę i dopiero po pewnym czasie uruchomić procedurę. App Inventor nie ma bloku **poczekaj**, musimy ten mechanizm stworzyć sami. W tym celu powracamy do ekranu **Designer**.



1 Niezbędne okaże się dodanie jeszcze jednego komponentu – **Clock** z kategorii **Sensors**.

2 Po przeciągnięciu go na okno podglądu gry nie zostanie on wyświetlony w oknie,

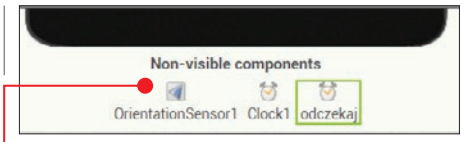
ale pod nim, jako niewidoczny komponent. Mimo że jest niewidoczny, daje nam do dyspozycji nowy zestaw bloków do wykorzystania w układaniu skryptów. Jednak zanim przejdziemy do bloków, możemy zmienić nazwę tego komponentu na zgodną z jego przeznaczeniem – czyli „odczekaj". Należy też nieco zmienić jego właściwości.



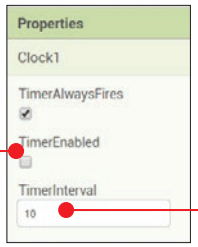
3 Właściwość **TimerEnabled** nie powinna być zaznaczona, by komponent odliczał czas nie od startu programu, ale od włączenia go z poziomu skryptów.

4 Właściwość **TimerInterval** powinna mieć wartość 300, by komponent odliczył 0,03 sekundy, a nie całą sekundę (wpisujemy czas wyrażony w milisekundach).

Jeśli jesteśmy już przy dodawaniu niewidocznych komponentów, potrzebne będą jeszcze dwa. Komponent **Clock** wykorzystamy jako „pętlę" do ciągłego sprawdzania, czy samochód znajduje się na torze jazdy, a **OrientationSensor** (również w kategorii **Sensors**) do sterowania samochodem poprzez pochylanie urządzenia.



1 W kolejnym Clocku także właściwość **TimerEnabled** nie powinna być zaznaczona, by komponent odliczał czas nie od startu programu, ale od włączenia go z poziomu skryptów.

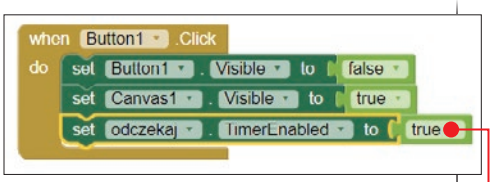


2 Właściwość **TimerInterval** powinna mieć wartość **10**, by komponent mógł wykonywać „pętlę" możliwie często.

3 Właściwości komponentu **OrientationSensor** z poziomu okna **Designer** powinny pozostać niezmienione.

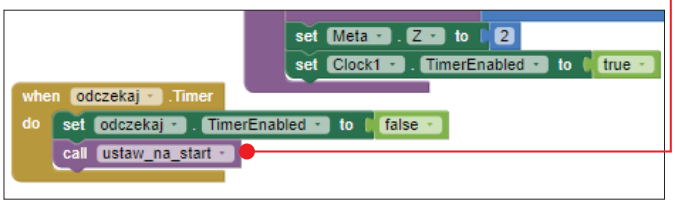
Wracamy do bloków.

1 Teraz musimy zająć się uruchomieniem „odczekania" – blok **set odczekaj.TimerEnabled to** powinien wykonywać się w momencie naciśnięcia przycisku, i włączać ten



komponent, czyli powinna być przyczepiona do niego wartość true.

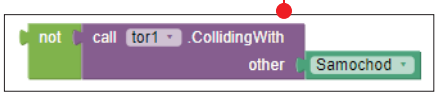
2 Z kategorii **odczekaj** wkładamy blok **when odczekaj.Timer do**. To co do niego włożymy, wykona się 0,03 s po wciśnięciu przycisku – wtedy powinna zostać uruchomiona ułożona wcześniej procedura. Wkładamy do tego bloku blok **call ustaw_na_start**. Oprócz tego wyłączamy ten Clock, by nie wykonywał on procedury ciągle, ale tylko raz.



3 Dodatkowo na końcu procedury **ustaw_na_Start** dodajemy instrukcję, która uruchomi **Clock1**, czyli zegar, który co 0,01 s będzie wykonywał swoje zdarzenie **.Timer()**. Co w nim ułożymy – to za chwilę, najpierw zajmijmy się sterowaniem samochodem.

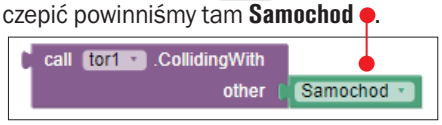
STEROWANIE SAMOCHODEM

Sterowanie samochodem zrobimy dzięki dodaniu komponentu **OrientationSensor**. Ma on w sobie blok zdarzenia **.OrintetionChanged** – przenosimy go na obszar roboczy. Wewnątrz niego należy ustawić kierunek (Heading) i prędkość (Speed) samochodu. Powinna ona być ustawiona na podstawie odczytów z **OrientationSensora** w sposób przedstawiony na rysunku.



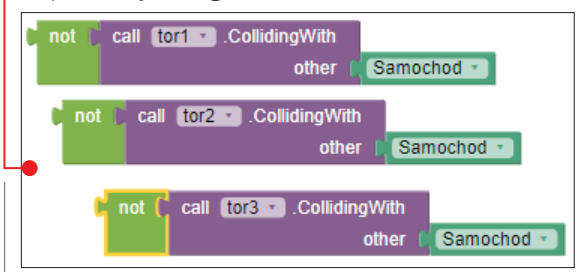
Samochód powinien wracać na start, jeśli nie dotyka toru. Ponieważ nasz tor podzielony jest na trzy obiekty – samochód nie dotyka „toru", gdy nie dotyka obiektu **tor1**, i nie dotyka obiektu **tor2**, i nie dotyka obiektu **tor3** – postarajmy się ułożyć to zadanie z bloków.

1 Blok **call tor1.CollidingWith other** zwraca prawdę, gdy tor1 koliduje z obiektem, który doczepimy do pola **other** w tym bloku. Doczepić powinniśmy tam **Samochod**.

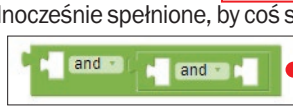


2 Ponieważ nas interesuje moment, gdy kolizji między obiektami już nie ma – z przodu doczepiamy zaprzeczenie – czyli blok **not** z kategorii **Logic**.

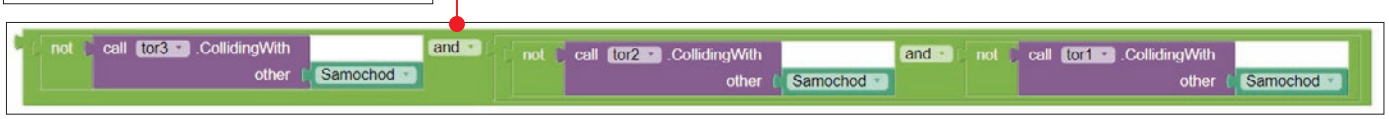
3 Analogiczny skrypt układamy dla dwóch pozostałych fragmentów toru.



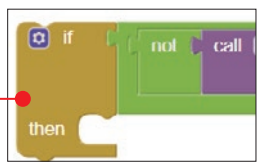
4 Z kategorii **Logic** bierzemy też dwa spójniki and – wkładamy jeden w drugi – w ten sposób dostaniemy blok, który pozwoli na połączenie trzech warunków – które muszą być jednocześnie spełnione, by coś się stało.



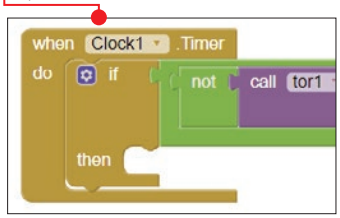
5 Wkładamy trzy bloki utworzone w kroku **3** do slotów w konstrukcji z kroku **4**.



6 Całą konstrukcję dołączamy do bloku **if** z kategorii **kontrol**. If to instrukcja warunkowa – to co włożymy do jej środka, wykona się, gdy warunek przypięty u góry będzie spełniony.



7 If wkładamy w blok **when Clock1.Timer do**, by warunek był sprawdzany z każdym wywołaniem zdarzenia Timera, czyli co 0,01 s.



8 Gdy warunek z if będzie spełniony, powinna wykonać się procedura **ustaw_na_start**, by samochód wrócił na pozycję startową.

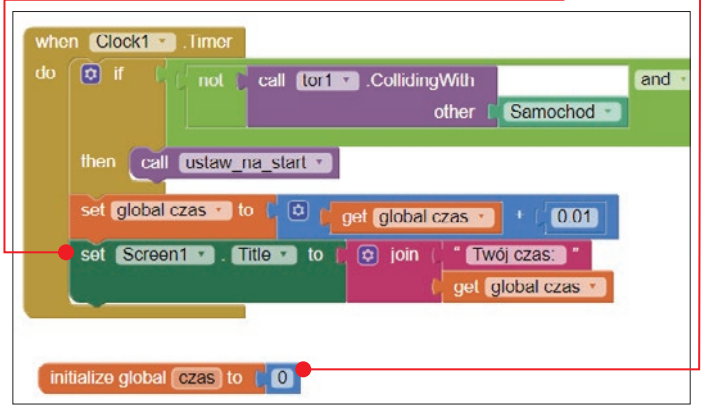
ODLICZANIE CZASU GRY

O jakości przeprowadzonej rozgrywki powinien decydować czas dojechania do mety.

1 Tworzymy zmienną o nazwie **czas** (w zmiennych możemy przechowywać różne wartości). W tym celu bierzemy pomarańczowy blok z kategorii **Variables** i wybieramy

korzystujemy go do stworzenia zmiennej **czas** o wartości początkowej **0**.

2 Z każdym przejściem pętli należy zwiększać wartość zmiennej **czas** o **0,01** i wyświetlać ją na pasku tytułowym, który widoczny jest nad **Canvas** podczas gry. Zrobimy to z wykorzystaniem bloku **set Screen1.Title to**.

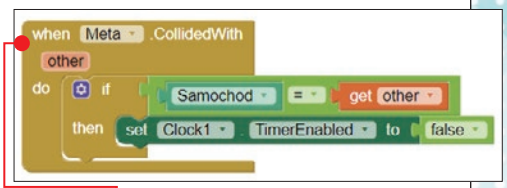


Sugestie

Dobrym sposobem na doskonalenie swoich umiejętności jest praca samodzielna – już bez instrukcji. Spróbujmy rozbudować grę o własne pomysły. Jeśli nam ich brakuje, oto kilka sugestii:

- Dodajmy przycisk do restartu gry, który pojawi się, gdy samochód dojedzie do mety i zniknie po kliknięciu, pozwalając na przeprowadzenie kolejnej rundy.
- Dodajmy nowe fragmenty toru – wprowadzając więcej zakrętów, które utrudnią szybkie przejechanie całej trasy.
- Dodajmy drugi poziom gry, który będzie pojawiał się, gdy samochód dojedzie do mety.

ZATRZYMANIE CZASU NA MECIE

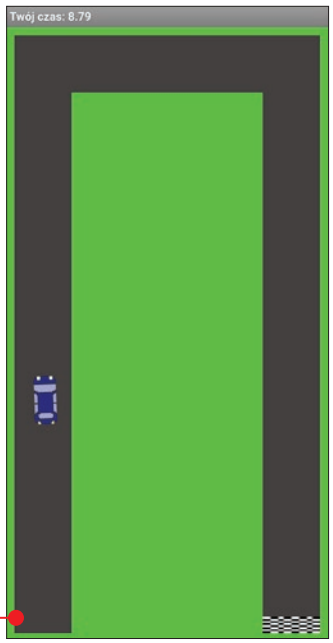


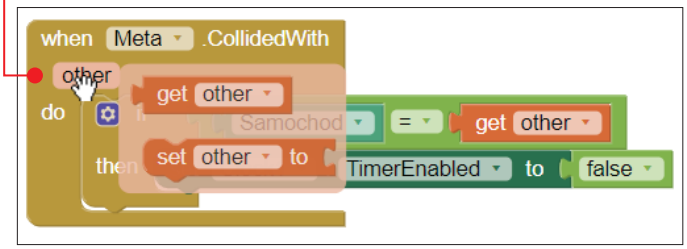
Na koniec powinniśmy zaprogramować to, co ma się dziać, gdy samochód dojedzie do mety.

1 Z kategorii **Meta** bierzemy blok **when Meta.CollidedWith other do**, i wkładamy do niego nowy if.

2 W if przypinamy warunek, że samochód jest tym obiektem, z którym koliduje **Meta**. Informacja o obiekcie jest ukryta w bloku **other**, wyświetlanym, gdy najedziemy kursorem na pole **other**.

3 Gdy warunek jest spełniony, należy wyłączyć wykonywanie pętli przez komponent **Clock1**. Pamiętajmy o tym, by na bieżąco testować działanie projektu. Efekt powinien wyglądać tak.



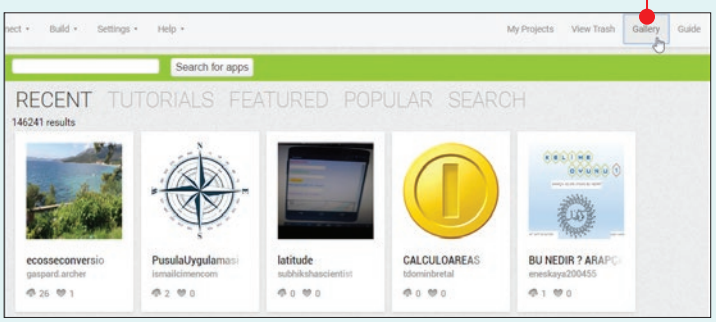


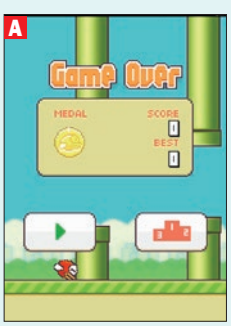
Galeria projektów

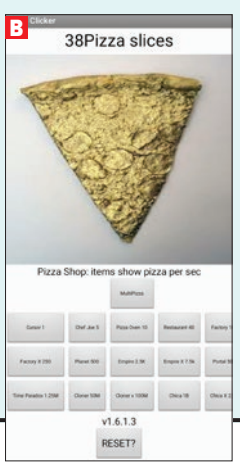
App Inventor pozwala na publikację swoich projektów. Dzięki temu powstaje galeria. W niej także można poszukać inspiracji do własnych gier. Aby wejść do galerii, należy kliknąć na przycisk **Gallery** na pasku menu głównego. W galerii możemy znaleźć między innymi:

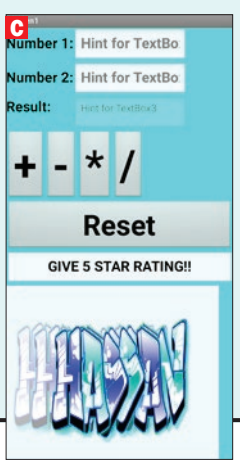
- klony popularnych gier – na przykład **Flappy Bird** **A**;
- gry typu kliker – na przykład **Pizza Clicker** **B**,
- ale też programy – jak stoper, symulator kostki do gry czy kalkulator **C**.

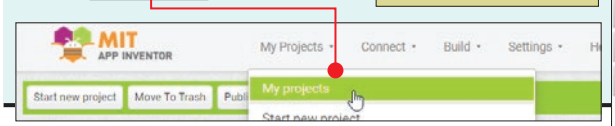
To tylko kilka przykładów projektów, jakie można tam znaleźć. Nasze projekty również mogą trafić do galerii. Rozwijamy opcję **My projects** z menu głównego i wybieramy pozycję **My projects**. W ten

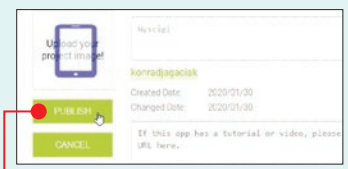




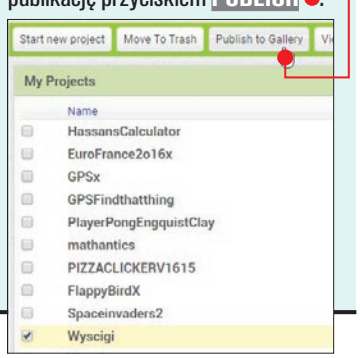








sposób dostaniemy się do zestawiania wszystkich projektów zrealizowanych na koncie, poprzez które logujemy się do programu. Zaznaczamy projekt, który chcemy udostępnić, i klikamy na przycisk **Publish to Gallery**. W kolejnym oknie uzupełniamy informacje o projekcie i zatwierdzamy publikację przyciskiem **PUBLISH**.



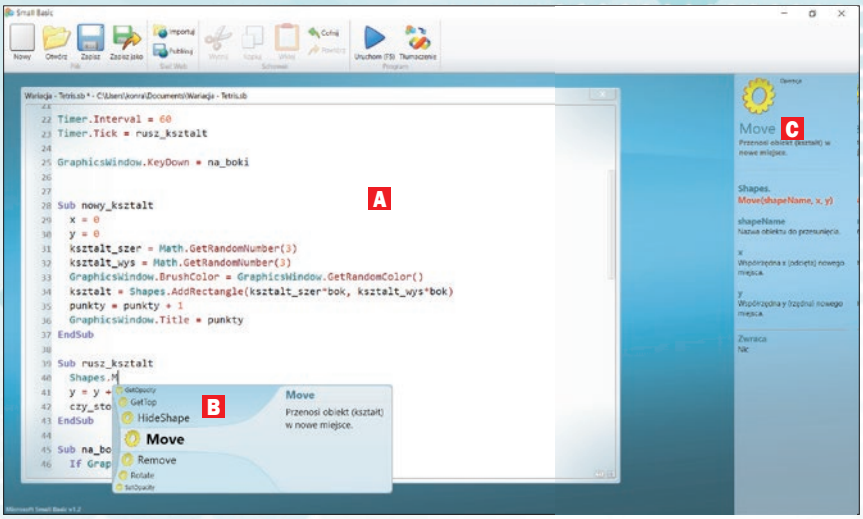
MICROSOFT SMALL BASIC: TETRIS

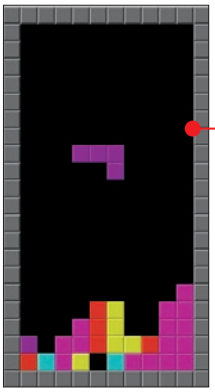
Microsoft Small Basic (DVD-KOD: **071**) to język pisany. Ma jednak uproszczoną składnię – został stworzony jako narzędzie edukacyjne, więc doskonale się sprawdza jako wprowadzenie do „poważniejszych" języków programowania.

OKNO PROGRAMU

Okno programu Microsoft Small Basic jest proste. Pomaga to użytkownikowi skupić się na tym, co w wykorzystaniu tego programu najważniejsze, czyli na tworzeniu skryptów. Całość aplikacji powstaje poprzez stworzenie kodu źródłowego w edytorze **A**, który stanowi główną część okna.

Edytor jest dość wygodny. Podczas pisania wyświetlane są sugerowane polecenia **B**. By skorzystać z sugestii i nie pisać ręcznie całego polecenia, wystarczy wcisnąć enter. Sugestia zawiera też skrócony opis działania poleceń. Natomiast pełny opis działania znajduje się na panelu podpowiedzi **C** po prawej stronie okna.





NOWA WERSJA TETRISA

Klasyczna gra Tetris została stworzona przez Aleksieja Pażytnowa w 1984 roku. Polega ona na przesuwaniu i obracaniu spadających z góry okna kształtów złożonych z kwadratowych klocków. Trzeba to robić w taki sposób, by poszczególne kwadraty układały się w jednej poziomej linii. Pomysł ten okazał się bardzo grywalny, dzięki czemu powstało wiele gier inspirowanych Tetrisem. W tej części kursu stworzymy właśnie mocno uproszczoną grę inspirowaną Tetrisem, jednak jej działanie i zasady będą nieco różniły się od oryginału. W naszej grze spadające kształty to prostokąty, które różnią się wielkością. Ich szerokość i wysokość może wynosić od jednej do trzech kratek. Zadaniem gracza jest takie przesuwanie prostokątów, by jak najwięcej zmieścić w oknie gry. Za każdy prostokąt gracz otrzyma jeden punkt. Wymiary prostokątów są losowe.

PRZYGOTOWANIE OKNA GRY

W oknie gry powinno się znaleźć kratkowane pole, na którym będzie widać kwadraty, z których zbudowane są wszystkie figury. Trzeba określić wymiary pola do gry, to znaczy ile kratek wysokości i szerokości będzie miało pole do prowadzenia rozgrywki. A te wymiary można zapisać w zmiennych.

Wariacja - Tetris.sb * - C:\Users\konra\Documents\Wariacja - Tetris.sb

```
1 szerokosc = 10
2 wysokosc = 20
```

1 W języku Small Basic zmienne tworzy się, po raz pierwszy używając ich nazw i przypisując im początkową wartość. Jeśli chcemy mieć zmienną o nazwie **szerokosc** i wartości początkowej **10**, wystarczy napisać **szerokosc = 10**. Podobnie można zrobić z wysokością. Tak jak w klasycznym Tetrisie powinna ona być większa niż szerokość, może wynosić 20. Zatem należy napisać **wysokosc = 20**.

2 Dwie utworzone zmienne określają, ile kratek ma być w oknie programu. Powinniśmy jeszcze określić wielkość jednej kratki. Dlatego poleceniem **bok = 30** tworzymy zmienną, której wartość określa wielkość boku jednego kwadratu (z takich kwadratów zbudowane będzie całe okno).

Wariacja - Tetris.sb * - C:\Users\konra\Document

```
1 szerokosc = 10
2 wysokosc = 20
3 bok = 30
```

3 Wielkość pojedynczego kwadratu ma wpływ na wymiary okna naszej gry. A wymiary okna powinny być takie, by zmieściło się całe pole gry. Zatem szerokość okna powinna być równa wartości zmiennej **bok** pomnożonej przez liczbę kwadratów, czyli wartość zmiennej **szerokosc**. By wynik tego mnożenia przypisać do wymiaru okna gry, piszemy **GraphicsWindow.Width = szerokosc*bok**.

```
szerokosc = 10
wysokosc = 20
bok = 30

GraphicsWindow.Width = szerokosc*bok
```

4 Analogicznie trzeba postąpić w wypadku wysokości. Właściwością odpowiadającą za wysokość okna jest **GraphicsWindow.Height**. Należy przypisać jej wartość będącą wynikiem mnożenia zmiennej **bok** i **wysokosc**.

```
GraphicsWindow.Width = szerokosc*bok
GraphicsWindow.Height = wysokosc*bok
```

5 Okno naszej gry powinna wypełnić kratka – tylko po polach kratki będzie można przesuwać prostokąty. Kratka powstanie w wyniku narysowania poszczególnych kwadratów, które będą ją tworzyć. Do rysowania prostokątów wykorzystuje się polecenie **GraphicsWindow.DrawRectangle**. Pozwala ono

w określonym miejscu (po podaniu odległości od lewej i górnej krawędzi okna) narysować prostokąt o określonych wymiarach. Gdy rysujemy kratkę, to zakładając, że mamy 10 kwadratów szerokości i 20 kwadratów wysokości, należałoby stworzyć 200 kwadratów. Nie oznacza to, że 200 razy trzeba użyć polecenia **GraphicsWindow.DrawRectangle** – wystarczy napisać je raz, ale jego wywołanie powinno następować w pętli.

6 By wykonać coś określoną liczbę razy, w programowaniu stosuje się pętlę **for**. I tak, jeśli chcemy narysować jedną kolumnę kwadratów, pętla powinna wykonać się tyle razy, ile jest kwadratów w pionie (o tym mówi zmienna **wysokosc**). Taką pętlę tworzymy poleceniem **For j = 1 To wysokosc**,

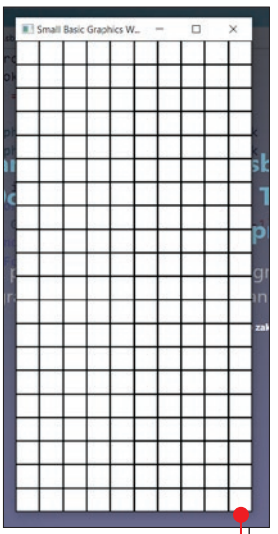
```
6  GraphicsWindow.Height = wysokc
7
8  For j = 1 To wysokosc
9
10 EndFor
```

a dodatkowo dodajemy linijkę **EndFor**, która wskazuje na zamknięcie pętli. To, co znajduje się pomiędzy otwarciem a zamknięciem, to treść pętli, czyli polecenia, które wykonają się określoną liczbę razy. Ile razy? Jeśli zmienna **wysokosc** ma wartość 20, to tyle razy wykona się pętla. Dodatkowo w pętli będzie zmienna **j**. Z każdym wykonaniem pętli będzie ona zwiększała swoją wartość o jeden, aż osiągnie wartość zmiennej **wysokosc**.

7 Aby narysować całą kratkę, rysowanie kwadratu trzeba zapętlić nie tylko w pionie, ale i w poziomie – takie kolumny powinny pojawić się też na całej szerokości okna gry. W tym celu trzeba użyć pętli w pętli. Tę pętlę odnoszącą się do jednej kolumny umieszczamy wewnątrz pętli, która wykona się dla każdego elementu w szerokości kratki. W ten sposób uzyskamy dwie pętle. To, co znajdzie się w pętli wewnętrznej, wykona się tyle razy, ile wynosi wynik mnożenia wartości zmiennych **szerokosc** i **wysokosc**, czyli 200.

```
For i = 1 To szerokosc
  For j = 1 To wysokosc

  EndFor
EndFor
```

8 Poszczególne kwadraty można rysować poleceniem **GraphicsWindow.DrawRectangle((i-1)*bok, (j-1)*bok, bok, bok)**. Zmienna **i** rośnie z każdym przejściem pętli zewnętrznej, a działanie **(i-1)*bok** pozwala obliczyć odległość danego kwadratu od lewej krawędzi okna. Podobnie ze zmienną **j** – rośnie ona z każdym przejściem pętli wewnętrznej, a wykorzystać ją można w działaniu **(j-1)*bok** do obliczenia odległości danego kwadratu od górnej krawędzi okna. Pozostałe dwa parametry, czyli **bok**, **bok**, odnoszą się odpowiednio do szerokości i wysokości prostokąta, które w tym przypadku są takie same, ponieważ rysujemy kwadrat.




```
8  For i = 1 To szerokosc
9    For j = 1 To wysokosc
10     GraphicsWindow.DrawRectangle((i-1)*bok, (j-1)*bok, bok, bok)
11   EndFor
12 EndFor
```

9 Naciskamy klawisz F5, by uruchomić stworzony do tej pory skrypt. Wyświetli się wówczas okno gry o określonych wcześniej wymiarach w całości wypełnione kratką.

RYSOWANIE SPADAJĄCYCH KSZTAŁTÓW I LICZENIE PUNKTÓW

Kolejnym krokiem będzie stworzenie mechanizmu generowania prostokątnych kształtów. Nowy kształt powinien być generowany za każdym razem, gdy poprzedni zatrzyma się na dole okna lub na innym kształcie, zatem odpowiedzialny za to fragment kodu będzie wykonywany wielokrotnie. Takie fragmenty kodu, które przeznaczone są do wielokrotnego użytku, zapisuje się jako procedury.

```
Sub nowy_ksztalt
  x = 0
  y = 0
A ksztalt_szer = Math.GetRandomNumber(3)
B ksztalt_wys = Math.GetRandomNumber(3)
C GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
D ksztalt = Shapes.AddRectangle(ksztalt_szer*bok, ksztalt_wys*bok)
E punkty = punkty + 1
F GraphicsWindow.Title = punkty
EndSub
```

1 Procedurę tworzymy, używając słowa kluczowego **Sub**, a dalej podając jej nazwę i poniżej umieszczając **EndSub**, by wskazać, gdzie kończy się definicja. Zatem by stworzyć procedurę odpowiedzialną za generowanie nowych kształtów, piszemy **Sub nowy_ksztalt** i poniżej **EndSub**.

```
Sub nowy_ksztalt

EndSub
```

2 Wewnątrz tej procedury należy utworzyć dwie zmienne **x** i **y** wskazujące na lokalizację kształtu w oknie gry. Jeśli będą miały wartości 0, to kształt znajdzie się idealnie w lewym górnym rogu okna, bo jego odległość od bocznej i górnej krawędzi wyniesie właśnie 0 pikseli.

```
Sub nowy_ksztalt
  x = 0
  y = 0
```

3 Całość procedury powinna mieć następującą postać, gdzie:

A Tworzy zmienną **ksztalt_szer**, której nadaje wygenerowaną wartość pseudolosową z zakresu od 1 do 3. Generowane liczby zawsze zaczynają się od 1 jako minimum, a maksimum może być określone przez programistę.

B Podobnie jak wyżej – tworzy zmienną **ksztalt_wys**, która dostanie wygenerowaną wartość od 1 do 3.

C Ustawia kolor wypełnienia kształtu na losowy.

D Tworzy obiekt o nazwie**ksztalt**, będący prostokątem o wymiarach odpowiadających liczbie wygenerowanych kratek. To inny sposób tworzenia figur niż w wypadku rysowania kratki. Takie obiekty, które tworzymy za pomocą słowa kluczowego **Shapes**, można modyfikować, na przykład przemieszczając je.

E Z każdym wygenerowanym, pojawiającym się w oknie gry kształtem dodawany jest jeden punkt do zmiennej **punkty**.

F Aktualna liczba punktów wyświetlana jest na pasku tytułowym.

4 By wywołać rysowanie kształtu, piszemy nazwę procedury z nawiasem na końcu – jeszcze przed definicją tej procedury. Ponieważ procedura w swojej treści odnosi się do zmiennej **punkty** i zwiększa o jeden jej wartość, to w momencie wywołania procedury zmienna ta powinna już istnieć. Piszemy **punkty = 0** przed wywołaniem procedury.

```
10     GraphicsWindo
11   EndFor
12 EndFor
13
14 punkty = 0
15 nowy_ksztalt()
16
17 Sub nowy_ksztalt
```

Wyrwij, użyj dziurkacza i zbieraj do segregatora

GRAWITACJA: SPRAWIAMY, BY FIGURY SPADAŁY

Co pewien określony czas kształt powinien obniżać swoją pozycję w oknie gry. A żeby wykonywać coś co pewien określony czas, należy stworzyć procedurę i przypisać ją do zdarzenia **Timer**, które jest wywoływane z określonym interwałem.

1 Interwał ten zapisujemy jako **Timer.Interval = 200**, gdzie **200** to liczba milisekund pomiędzy kolejnymi wywołaniami procedury.

```
Timer.Interval = 200
Timer.Tick = rusz_ksztalt
```

2 Samą procedurę przypisujemy do zdarzenia poleceniem **Timer.Tick = rusz_ksztalt**.

3 Poniżej należy jednak zdefiniować procedurę o użytej nazwie. Może ona mieć postać, gdzie:

```
Sub rusz_ksztalt
A Shapes.Move(ksztalt,x,y)
B y = y + bok
C czy_stoi()
EndSub
```

A **Shapes.Move** to polecenie pozwalające na przeniesienie obiektu w miejsce określone współrzędnymi **x** i **y**.

B Zwiększa odległość kształtu od górnej krawędzi, poprzez zwiększenie wartości zmiennej **y**.

C Uruchamia procedurę o nazwie **czy_stoi**, której zadaniem jest sprawdzenie, czy poruszający się w dół kształt powinien się już zatrzymać.

ZATRZYMANIE RUCHU KSZTAŁTU

Aby spadające kształty zatrzymywały się, należy wyznaczyć granicę, na której powinny stawać. Granica ta musi być różna na każdej kratce.

1 Wartości granic dla poszczególnych kratek można przechowywać w tablicy. Tablica powinna mieć indeksy numerowane od 1 do wartości zmiennej **szerokosc**, bo tyle granic będzie do zapisania.

2 Można użyć pętli **for**, w której zmienna pomocnicza posłuży jako wyznaczenie indeksu dla kolejnych elementów tablicy.

3 Wszystkie elementy tablicy powinny mieć jednakową wartość odpowiadającą wysokości okna gry, bo to tam mają zatrzymywać się na początku spadające kształty. Całość możemy zapisać następująco.

```
14 punkty = 0
15 nowy_ksztalt()
16
17 For z = 1 To szerokosc
18   granica[z] = GraphicsWindow.Height
19 EndFor
```

4 Stworzoną w ten sposób tablicę należy wykorzystać w treści procedury **czy_stoi**, którą trzeba zdefiniować, aby odnosząca się do niej procedura **ruszkształt** działała prawidłowo. Na końcu skryptu piszemy zatem **Sub czy_stoi EndSub**.

```
Sub czy_stoi
A For z = 1 To szerokosc
  B For q = 1 To ksztalt_szer
    C If y+(ksztalt_wys*bok) > granica[z] And (x/bok)+q = z Then
      D For w = 1 To ksztalt_szer
        E granica[(x/bok)+w] = y-bok
      F EndFor
      G czy_nowy()
    H EndIf
  I EndFor
J EndFor
EndSub
```

```
Sub czy_stoi

EndSub
```

5 Całość procedury może mieć następującą formę, gdzie:

A Pętla **for** pozwala na odniesienie się do numerów poszczególnych kratek na szerokości okna.

B Pętla **for** pozwala na odniesienie się do poszczególnych kratek na szerokości kształtu.

C **Instrukcja warunkowa** sprawdza, czy dolna krawędź spadającego kształtu przekracza wartość granicy w kratce odpowiadającej lokalizacji poszczególnych kratek na szerokości spadającego kształtu.

D Kolejna pętla **for** pozwala na odniesienie się do poszczególnych kratek na szerokości kształtu.

E Ustawia granicę na kratkach odpowiadających tym, na których znajduje się kształt, tak by kolejne kształty, spadając, zatrzymywały się już na nowej granicy.

F Zamyka pętlę **for** z punktu **D**.

G Uruchamia procedurę **czy_nowy** – nie jest ona jeszcze zdefiniowana, ale jej działanie będzie polegało na sprawdzeniu, czy powinien pojawić się nowy kształt.

H Zamyka **instrukcję warunkową** z punktu **C**.

I Zamyka pętlę **for** z punktu **B**.

J Zamyka pętlę **for** z punktu **A**.

SPADANIE NOWEGO KSZTAŁTU

By w oknie gry pojawił się nowy kształt, wystarczy uruchomić procedurę **nowy_ksztalt**, którą mamy już zdefiniowaną. Jednak nie wystarczy zaprogramować, by nowy kształt pojawiał się, gdy poprzedni osiągnie granicę. Najpierw trzeba sprawdzić, czy jest jeszcze miejsce na nowe kształty, czyli czy ułożone już prostokąty nie mają granicy na samej górze okna. I w tym właśnie celu należy stworzyć procedurę **czy_nowy**, która powinna mieć postać, gdzie:

```
Sub czy_nowy
A For b = 1 To szerokosc
  B If granica[b] <= 0 Then
    C Timer.Tick = stop
  D EndIf
E EndFor
F nowy_ksztalt()
EndSub
```

A Otwiera pętlę **for**, która pozwoli odnieść się do wszystkich kratek na szerokości okna.

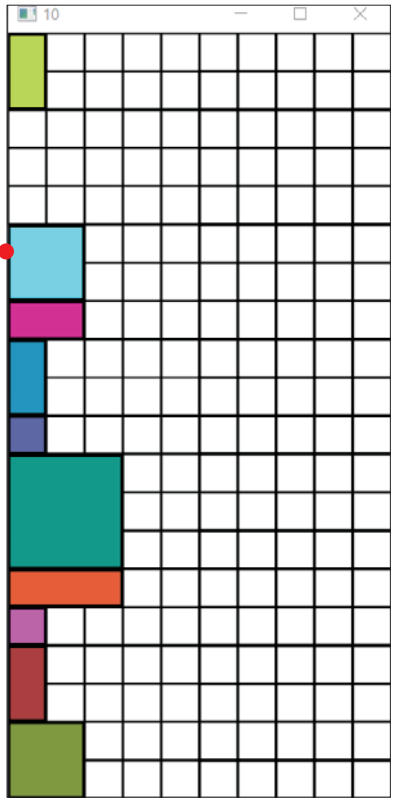
B Tworzy instrukcję warunkową, która sprawdza, czy granica na kolejnych kratkach jest już równa zeru lub mniejsza od zera.
C Do zdarzenia **Timer.Tick** przypisuje procedurę **stop** – należy ją zdefiniować pod omawianą procedurą. Procedura **stop** powinna być pusta, dzięki czemu nic nie będzie się działo przy zdarzeniu **Tick** dla Timera – nowy kształt przestanie spadać, a kolejne nie będą się pojawiać.

```
Sub stop
EndSub
```

D Zamyka **instrukcję warunkową** z punktu **B**.
E Zamyka pętlę **for** z punktu **A**.
F Uruchamia procedurę **nowy_ksztalt**.

STEROWANIE KSZTAŁTEM NA BOKI

Na tym etapie mamy już aplikację, w której prostokąty spadają z góry okna. Jednak wszystkie spadają w taki sposób, że nie poruszają się na boki. To jest właśnie zadanie gracza – poruszać kształtami, korzystając ze strzałek.



1 By coś działo się w momencie wciśnięcia jakiegoś klawisza na klawiaturze, musi być przypisane do zdarzenia **GraphicsWindow.KeyDown**. Jeśli chcemy, by procedura o nazwie **na_boki** wykonywała się wraz z wciśnięciem klawisza, napiszmy **GraphicsWindow.KeyDown = na_boki**.

```
GraphicsWindow.KeyDown = na_boki
```

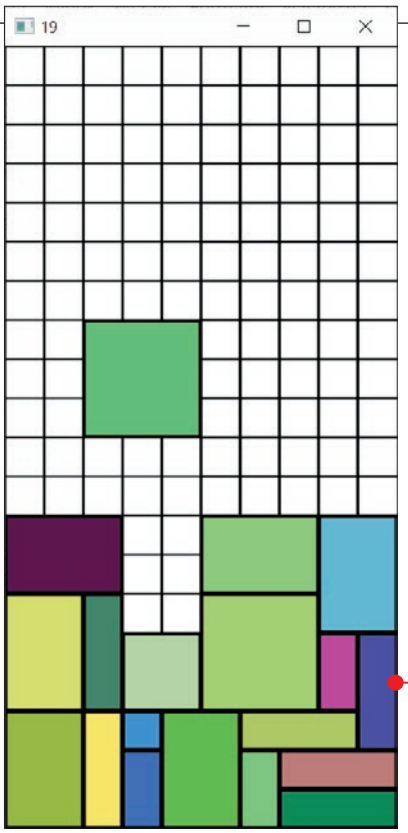
2 Następnie trzeba zacząć tworzyć definicję procedury **na_boki**.

```
GraphicsWindow.KeyDown = na_boki

Sub na_boki

EndSub
```

3 W procedurze trzeba stworzyć instrukcję warunkową, gdzie warunkiem jest to, że ostatnio wciśnięty klawisz (ta informacja jest przechowywana jako **GraphicsWindow.LastKey** **A**) to strzałka w lewą stronę, która jest przez Small Basic rozpoznawana jako **"Left"** **B**. Dodatkowo w instrukcji powinien być spełniony drugi warunek: odległość kształtu od lewej krawędzi okna musi być większa niż zero **C**, tak by kształt nie „wyszedł" poza okno. Gdy oba warunki (łączymy je spójnikiem **And** **D**) będą spełnione, należy zmniejszyć odległość kształtu od lewej krawędzi o wielkość jednego boku kratki. Zapisujemy to jako **x = x - bok**.

```
Sub na_boki
  If GraphicsWindow.LastKey = "Left" And x>0 Then
    x = x - bok
  EndIf
```

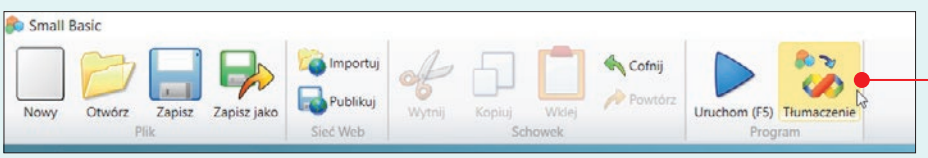
4 Analogicznie – trzeba zrobić instrukcję warunkową, która sprawdzi, czy ostatnio wciśniętym klawiszem (**GraphicsWindow.LastKey**) była strzałka w prawą stronę (**"Right"** **E**) i czy jednocześnie cały kształt znajduje się w oknie gry – tak by nie dało się „wyjść" z okna także przez jego prawą krawędź. Na koniec przetestujmy grę i spróbujmy umieścić w oknie jak najwięcej prostokątów, zanim gra się zatrzyma.




```
Sub na_boki
  If GraphicsWindow.LastKey = "Left" And x>0 Then
    x = x - bok
  EndIf
  If GraphicsWindow.LastKey = "Right" And x+ksztalt_szer*bok < GraphicsWindow.Width Then
    x = x + bok
  EndIf
EndSub
```

Small Basic a Visual Basic.NET

Small Basic jest językiem, którego składnia czerpie z BASIC-a – języka programowania wysokiego poziomu, który został opracowany w latach sześćdziesiątych XX wieku. Dał on też początek językowi Visual Basic, który z kolei powstał w la tach dziewięćdziesiątych XX wieku. Wraz z pojawieniem się platformy .NET ukazała się też nowa wersja języka Visual Basic.NET (VB.NET). Jest ona rozwijana do dziś. Small Basic, który poznaliśmy w tym odcinku kursu, jest połączony właśnie z Visual Basic.NET. Znajomość Small Basica ułatwia na dalszym etapie nauki programowania poznawanie VB.NET. Warto też wiedzieć, że dzięki wbudowanym w aplikację mechanizmom można łatwo tłumaczyć skrypty Small Basica na język VB.NET. By przerobić swój skrypt na VB.NET, klikamy na przycisk **Tłumaczenie**. Z językiem VB.NET zapoznamy się w kolejnym odcinku kursu.



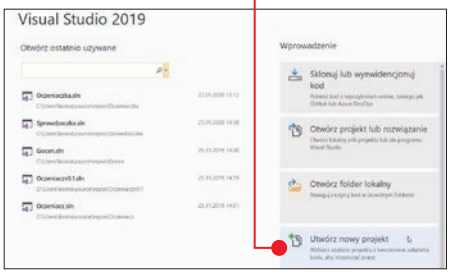
VISUAL BASIC .NET

WSTĘP DO PROFESJONALNYCH JĘZYKÓW PROGRAMOWANIA

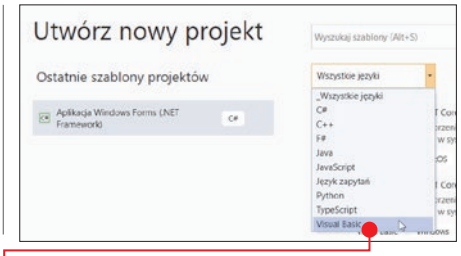
Trzecia część kursu programowania dotyczy profesjonalnego języka programowania, jakim jest **Visual Basic .NET**, w skrócie **VB .NET**. Poznamy podstawy jego składni i zobaczymy, jak napisać w nim prosty program. Skorzystamy w tym celu z narzędzia **Visual Studio 2019** (W KŚ+). **kj**

TWORZENIE NOWEGO PROJEKTU

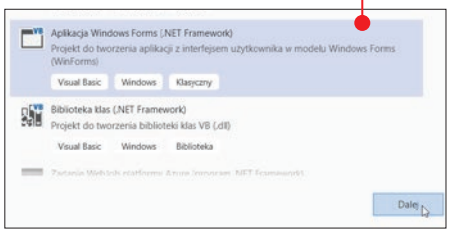
1 Po uruchomieniu Visual Studio 2019 można przejść do tworzenia nowego projektu. W tym celu klikamy na kafelek **Utwórz nowy projekt**.



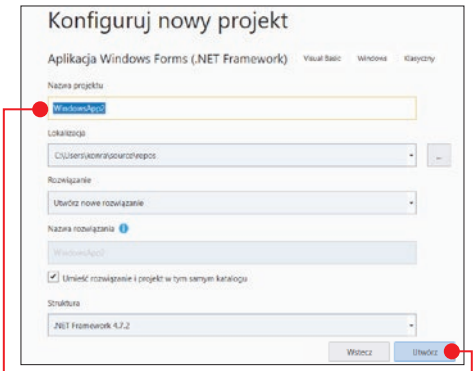
2 Projekty w Visual Studio tworzymy na podstawie szablonów. A szablony są podzielone na kategorie. Ponieważ Visual Studio pozwala na programowanie w wielu językach – język też jest jedną z kategorii podziału szablonów. Trzeba określić, w jakim języku ma być pisany tworzony projekt. W naszym przykładzie wybieramy język **Visual Basic** z rozwijanej listy.



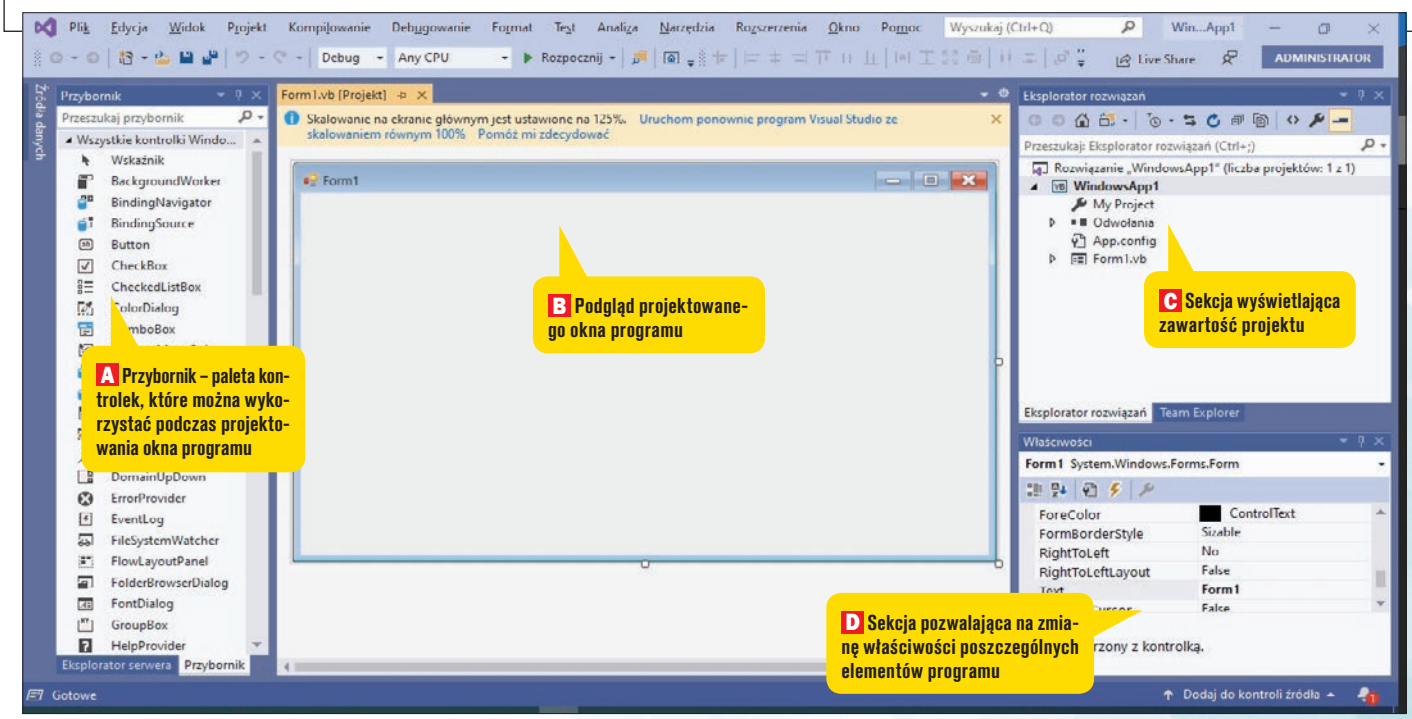
3 Teraz na liście szablonów będą znajdować się już tylko te dotyczące wskazanego przez nas języka. Spośród nich, na potrzeby realizacji kolejnych punktów naszych wskazówek, należy wybrać **Aplikacja Windows Forms (.NET Framework)** i zatwierdzić wybór, klikając na przycisk **Dalej**.



4 Kolejny krok pozwala na nadanie nazwy tworzonemu projektowi. W tym kroku można też zmienić lokalizację, w jakiej ma być on utworzony. Gdy uzupełnimy te dane, możemy kliknąć na przycisk **Utwórz**, by kreator wygenerował projekt.



5 Po utworzeniu nowy projekt jest jeszcze pusty, ale zawiera już okno tworzonego właśnie programu. Można teraz zaprojektować wygląd aplikacji, korzystając z przybornika **A**, który jest w panelu po lewej stronie. Są w nim elementy takie jak przyciski czy pola tekstowe. Przeciągając je na obszar modelu okna **B**, możemy je dodawać do aplikacji. Po prawej stronie okna Visual Studio widać sekcję **Eksplorator rozwiązań C** – w niej pokazana jest zawartość projektu, a jednym z jego elementów jest okno programu, i sekcję **Właściwości D** – do modyfikowania elementów programu.

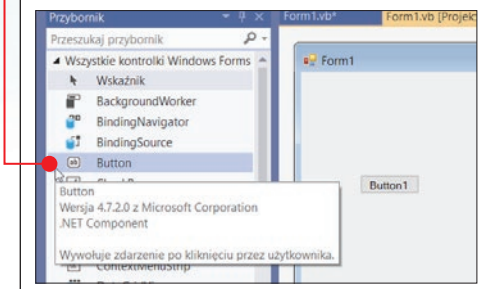


ZADANIE: GRA W ZAGADKI

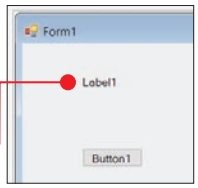
Zadaniem do zrealizowania w tym odcinku kursu będzie stworzenie prostej gry zawierającej pulę zagadek. Gra będzie wyświetlała zagadki, a zadaniem gracza będzie udzielanie prawidłowych odpowiedzi. Każda zagadka będzie zawierała podpowiedź, czyli zdanie wskazujące, co może być odpowiedzią, a także zawierające informacje o liczbie liter w poszukiwanym słowie i o tym, jaka jest jego pierwsza i ostatnia litera.

MODELOWANIE OKNA GRY

1 Aby zamodelować okno naszej gry, przeciągamy z przybornika potrzebne elementy – jednym z nich jest przycisk do zatwierdzenia odpowiedzi, czyli **Button**.

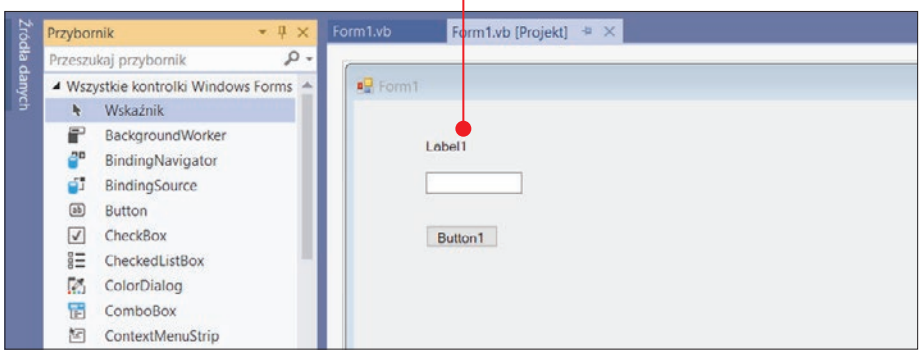


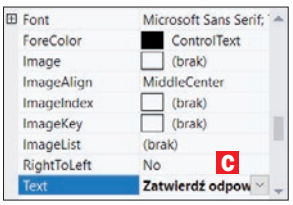
2 Kolejny element to etykieta do wyświetlania poszukiwanego słowa i treści zagadki – **Label**.

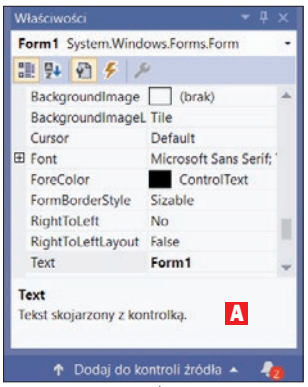


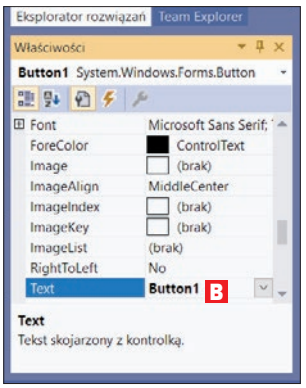
3 Ostatni niezbędny element to pole tekstowe do wpisania przez użytkownika odpowiedzi – **TextBox**.

4 Wszystkie kontrolki z przybornika można dostosować do własnych potrzeb,



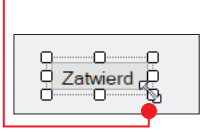


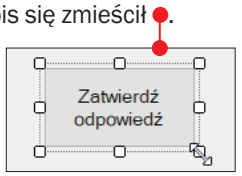




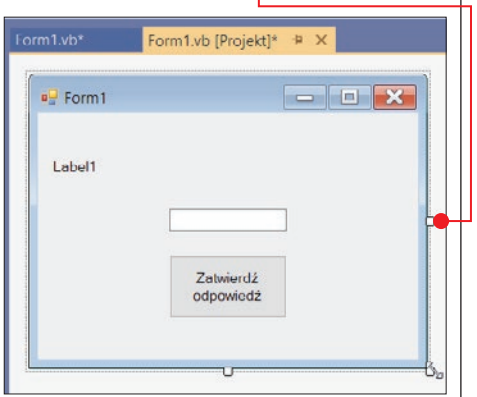
mieniając ich właściwości. Po zaznaczeniu kontrolki jej właściwości widoczne są w panelu w prawym dolnym rogu okna programu **A** (ten mechanizm znamy już z pierwszego odcinka naszego kursu dotyczącego App Inventora). Jedną z właściwości, które należy zmienić, jest napis na przycisku. Użytkownik naszej gry musi wiedzieć, do czego służy przycisk, a powinien o tym informować umieszczony na nim napis. Zatem zaznaczamy dodany do okna przycisk i w oknie **Właściwości** zmieniamy wartość dla właściwości **Text**: z wartości **Button1** **B** na przykład na **Zatwierdź odpowiedź** **C**.

5 Niektóre z właściwości można zmieniać też bezpośrednio w modelowanym oknie – do nich należy wielkość kontrolek. Zaznaczając kontrolkę i ustawiając kursor na jej krawędzi, zobaczymy strzałki pozwalające na rozciąganie i ściąganie kontrolki. Zauważmy, że teraz, po zmianie, napis na przycisku nie mieści się. Zaznaczamy przycisk i ustawiamy kursor na jego krawędzi, a potem klikamy – w ten sposób włączamy możliwość rozciągania przycisku – i rozciągamy go tak, by napis się zmieścił.





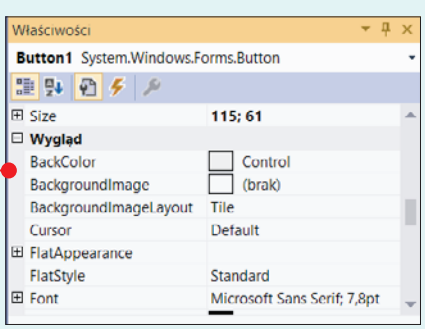
6 W ten sam sposób można zmienić też wielkość samego okna. Klikamy na wolny od kontrolek obszar tła modelowanego okna gry – gdy teraz ustawimy kursor na krawędzi modelu okna, będziemy mogli



zmienić jego rozmiar. Model domyślnie jest dość duży, zbyt duży na grę, która składa się tylko z trzech kontrolek. Zmniejszamy więc okno. Przed przejściem do pisania kodu źródłowego możemy jeszcze poustawiać kontrolki według uznania, tak by okno odpowiadało naszym preferencjom.

Zadbaj o wygląd

Przyciski, pola tekstowe, etykiety czy wszystkie inne elementy okna graficznego, a nawet samo okno graficzne, gdy dodajemy je do programu, mają swój domyślny wygląd. Nie oznacza to jednak, że muszą tak wyglądać również w ostatecznej wersji programu. W sekcji **Właściwości** znajdziemy wiele możliwości zmiany wyglądu elementów okna. Znajduje się tam nawet oddzielna grupa właściwości o nazwie **Wygląd**. Znajdują się tam właściwości pozwalające między innymi na ustawienie koloru tła (**BackColor**), czcionki (**Font**) czy koloru tekstu (**ForeColor**). Zestaw właściwości, które można zmieniać, jest różny w zależności od typu kontrolki, dla której są one ustawiane. Warto jednak poświęcić trochę czasu na eksperymenty z tymi ustawieniami, aby dopasować wygląd modelowanego okna do swoich potrzeb.

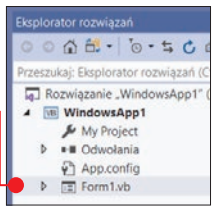


Wyrwij, użyj dziurkacza i zbieraj do segregatora

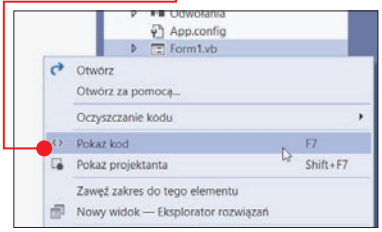
TWORZENIE SKRYPTU

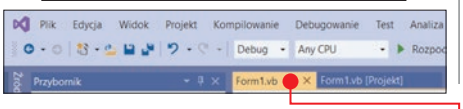
Za nami już sporo pracy, a żadna linijka kodu jeszcze nie została napisana. Już czas to zmienić.

1 W **Eksploratorze rozwiązań** klikamy prawym przyciskiem myszy na **Form1.vb**. To element projektu odpowiedzialny za istnienie okna gry.



2 Z menu kontekstowego wybieramy pozycję **Pokaż kod**.





3 Spowoduje to otwarcie drugiej zakładki projektu. Znajduje się w niej już początek skryptu. Możemy przełączać się pomiędzy zakładkami **Form1.vb** oraz **Form1.vb [Projekt]**, by przenosić się pomiędzy skryptem a modelem okna gry.

```
Public Class Form1

End Class
```

4 Skrypt, który już znajduje się w pliku, to zalążek klasy **Form1**. Klasa ta jest opisem całego okna.

Co to są klasy

Klasy to fundamentalny element programowania obiektowego – stanowią definicję obiektów. Opisują ich właściwości, a także funkcjonowanie. Tu obiektem jest konkretna instancja okna naszej gry, ale możemy tworzyć klasy opisujące dowolne zagadnienia i obiekty życia codziennego, które mają odzwierciedlenie w programie. Idea programowania obiektowego pozwala na znaczne skrócenie skryptów. Wyobraźmy sobie tworzenie gry piłkarskiej. Można w niej stworzyć definicję klasy o nazwie **Piłkarz**, która będzie opisywała, jakie właściwości ma ogólny piłkarz i jakie jest jego działanie – zależnie od wartości właściwości przypisanych do piłkarza. Następnie, tworząc już konkretnych zawodników, przypisuje się im jedynie wartości właściwości wymaganych do wygenerowania obiektu. Stworzenie każdego kolejnego piłkarza ogranicza się do jednej linii kodu tworzącej nowy obiekt. To ogromne ułatwienie programowania.

5 Wróćmy jednak do klasy **Form1**. Zastanówmy się, jakie są elementy logiki naszej gry, które będą nam niezbędne do jej zaprogramowania. Gra ma wyświetlać losowe zagadki. Zatem trzeba mieć pewną pulę zagadek, z której mogą być one losowane. Zagadka powinna się składać z podpowiedzi wskazującej na odpowiedź i słowa, które jest odpowiedzią. Zarówno podpowiedzi, jak i odpowiedzi możemy zapisać w tablicach. Czym są tablice, wiemy już z poprzedniego odcinka naszego kursu programowania. Pod jedną nazwą pozwalają przechowywać wiele wartości, które są ponumerowane indeksami. Tu potrzebne są dwie tablice – jedna na podpowiedzi, a druga na odpowiedzi. Elementy obydwu tablic powinny być ułożone w taki sposób, by jeden indeks dotyczył jednej zagadki – pierwsza podpowiedź w tablicy ma dotyczyć pierwszej odpowiedzi w tablicy itd.

Co to jest typ zmiennej

Zmienne mogą mieć różne typy. Na to, jakiego typu zmiennej potrzebujemy, ma wpływ jej zastosowanie.
Typy zmiennych można podzielić na trzy grupy – ze względu na ich wartości. W każdej z grup może znajdować się wiele typów, które różnią się zazwyczaj wielkością (liczbą bitów wykorzystaną do przechowania danych).

- Pierwsza z nich to **typy zmiennych liczbowych**. Tu możemy wydzielić dwie podgrupy – dla liczb całkowitych i zmiennoprzecinkowych. Takim typem dla liczb całkowitych jest **Integer** – wykorzystuje się go do zapisywania takich rzeczy, jak punkty czy życia w grach.
- Kolejna podgrupa to typy rzeczywiste pozwalające przechować liczby i ułamki. Tu można wymienić typy **Float** i **Double**. Możemy wykorzystać ten typ zmiennych na przykład do przechowywania temperatury ciała człowieka, która powinna wynosić 36,6 stopnia Celsjusza, czy choćby zapisu odległości uzyskanej przez lekkoatletę skaczącego w dal, gdzie odległość podaje się w metrach i centymetrach – na przykład 7,89.
Gdybyśmy chcieli przechować takie liczby w zmiennych typu **Integer**, wartość znajdująca się po przecinku nie zapisałaby się.
- Następna grupa to **typy znakowe** – do niej należy potrzebny nam w tej wskazówce typ **String**, który pozwala przechować ciąg znaków. Nawet jeśli w zmiennej typu **String** będziemy przechowywać liczby, będą one traktowane jak ciągi znaków. Czym to skutkuje?
Załóżmy, że mamy dwie zmienne typu **String** – **A** = „6" i **B** = „66". Jaki będzie wynik działania **A** + **B**? Nie będzie to „72", ale „666", ponieważ zostaną zestawione obok siebie dwa ciągi znaków, a nie dodane dwie liczby.
- Istnieje jeszcze typ **zmiennych logicznych** – **Boolean**, który pozwala na przechowywanie jednej z dwóch wartości – **True** lub **False**. Takich zmiennych możemy używać, kiedy przechowujemy informację odpowiadającą na pytanie „Czy...?", na które możemy udzielić odpowiedzi tak lub nie.
- Tablice, podobnie jak zmienne, też muszą mieć swój określony typ. Deklaracja tablicy różni się od deklaracji zmiennej jedynie tym, że po nazwie dodajemy pusty nawias – **()**.
Tworząc zmienną czy tablicę, możemy nadać jej od razu wartość początkową. Po typie danych dajemy znak równości i przypisujemy wartość. W przypadku tablic wartości umieszczamy w nawiasie klamrowym i oddzielamy je przecinkami. Wartości tekstowe zapisuje się w cudzysłowie.

```
Public Class Form1
    Dim slowa() As String = {"kolejka", "makaron", "kosmita"}
    Dim podpowiedzi() As String = {"Stoisz w niej w sklepie", "Na przykład spaghetti", "

End Class
```

6 Tworzenie zmiennych i tablic w VB. NET wygląda nieco inaczej niż w Small Basicu – tam wystarczyło pierwszy raz użyć zmiennej, by została ona utworzona. Tu bardziej przypomina to mechanizm znany nam już z App Inventora – zmienną trzeba zadeklarować, czyli należy użyć polecenia do jej tworzenia. Robi się to według schematu: **Dim nazwa_zmiennej As typ_zmiennej**. **Dim** jest tu słowem kluczowym. A czym jest typ zmiennej, dowiemy się z ramki powyżej.

7 Poleceniem **Dim slowa() As String = {"kolejka", "makaron", "kosmita"}** utworzymy tablicę o nazwie **slowa**, w której znajdą się trzy wartości – odpowiedzi do zagadek. Oczywiście w swojej grze możemy umieścić znacznie więcej słów niż w zaprezentowanym tu przykładzie.

8 Analogicznie do tablicy z odpowiedziami możemy utworzyć tablicę z podpowiedziami – **Dim podpowiedzi() As String = {"Stoisz w niej w sklepie", "Na przykład spaghetti", "Mieszka w kosmosie"}**. W tablicy muszą znaleźć się podpowiedzi do wszystkich odpowiedzi ułożone w tej samej kolejności.

9 Poleceniem **Dim wylosowane As String** utworzymy zmienną – przechowamy w niej wylosowane do rozgrywki słowo, które ma odgadnąć gracz.

```
1 odwołanie
Public Class Form1
    Dim slowa() As String = {"kolejka", "makaron", "ko
    Dim podpowiedzi() As String = {"Stoisz w niej w sk
    Dim wylosowane As String

End Class
```

```
Public Class Form1
    Dim slowa() As String = {"kolejka", "makaron", "kosmita"}
    Dim podpowiedzi() As String = {"Stoisz w niej w sklepie", "Na przykład spaghetti",
    Dim wylosowane As String

    Odwołania: 0
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    End Sub
End Class
```

10 By wylosować słowo, trzeba zrobić to, gdy okno gry już się załaduje. Załadowanie okna gry jest zdarzeniem, do którego można przypisać zdefiniowane zachowanie. By coś wydarzyło się w momencie załadowania się okna, przechodzimy do zakładki modelowania okna i klikamy dwukrotnie na pasek tytułowy projektowanego okna gry. Wygeneruje to metodę **Form1_Load**, jej treść wykona się po załadowaniu okna i to tam trzeba zapisać losowanie słowa.

11 Poleceniem **Dim r As Random = New Random()** tworzymy obiekt klasy **Random** pozwalający na wygenerowanie liczby.

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handl
    Dim r As Random = New Random()
    Dim numer As Integer = r.Next(slowa.Length)
    wylosowane = slowa(numer)
End Sub
```

12 Poleceniem **Dim numer As Integer = r.Next(slowa.Length)** utworzymy zmienną **numer**, będzie w niej przechowywany indeks, pod którym znajduje się wygenerowany numer podpowiedzi i odpowiedzi z zagadki. Polecenie **r.Next(slowa.Length)** generuje liczbę nieujemną, mniejszą od wartości podanej w nawiasie. A w nawiasie użyliśmy polecenia **slowa.Length**, które zwraca liczbę elementów tablicy o nazwie **slowa**. Elementy w tablicy mają indeksy ponumerowane od zera, więc maksymalny indeks elementu, jaki możemy wygenerować, jest o jeden mniejszy od liczby elementów tablicy – idealnie wpisuje się to w nasze zapotrzebowanie.

13 Linijka **wylosowane = slowa(numer)** pod nazwą **wylosowane** pozwala przechować jedno słowo z tablicy.

14 Dalej należy sprawić, by na etykiecie pojawiła się treść zagadki, a także pierwsza i ostatnia litera odpowiedzi, wszystkie pozostałe litery odpowiedzi powinny zaś być pokazane w formie kresek. Zatem dla słowa „makaron" powinniśmy pokazać „m _ _ _ _ _ n". Linia **Label1.Text = wylosowane(0)** ustawia tekst w etykiecie na pierwszą literę wybranego słowa.

```
Dim numer As Integer = r.Next(slow
wylosowane = slowa(numer)
Label1.Text = wylosowane(0)
```

15 Poprzez pętlę **For** do tekstu etykiety dodajemy tyle kresek, ile liter w słowie ma być „wykreskowanych".

```
Dim r As Random = New Random()
Dim numer As Integer = r.Next(slowa.Length)
wylosowane = slowa(numer)
Label1.Text = wylosowane(0)
For i As Integer = 1 To wylosowane.Length - 2
    Label1.Text += " _ "
Next
```

```
    For i As Integer = 1 To wylosowane.Length - 2
        Label1.Text += " _ "
    Next
    Label1.Text += wylosowane(wylosowane.Length - 1)
End Sub
```

16 Linią kodu **Label1.Text += wylosowane(wylosowane.Length - 1)** dodajemy do tekstu etykiety ostatnią literę słowa.

17 Na koniec dopisujemy jeszcze **Label1.Text += " " + podpowiedzi(numer)** – w ten sposób do tekstu etykiety dodana zostanie jeszcze treść podpowiedzi.

```
        Next
        Label1.Text += wylosowane(wylosowane.Length - 1)
        Label1.Text += "  " + podpowiedzi(numer)
    End Sub

    Odwołania: 0
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

    End Sub
End Class
```

18 Teraz można zająć się udzielaniem odpowiedzi przez gracza. Gracz powinien umieścić ją w polu tekstowym i kliknąć na przycisk, by sprawdzić, czy udzielona odpowiedź jest prawidłowa. Potrzebujemy zatem wykonać coś, gdy przycisk będzie kliknięty. Wracamy do zakładki projektowania okna gry i klikamy dwukrotnie na przycisk. Spowoduje to wygenerowanie metody, której treść będzie zawsze wykonywana po kliknięciu na przycisk przez gracza.

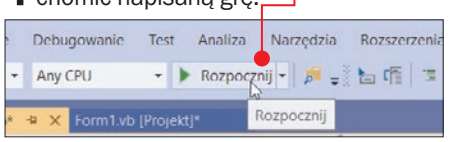
```
Private Sub Button1_Click(sender As Object, e A
    If wylosowane = TextBox1.Text Then
        MessageBox.Show("BRAWO! WYGRYWASZ")
    Else
        MessageBox.Show("Spróbuj ponownie.")
    End If
End Sub
```

19 W jej treści trzeba umieścić instrukcję warunkową **If**, gdzie warunkiem jest porównanie tekstu z pola tekstowego z wygenerowanym tekstem z bazy. Jeżeli są takie same, trzeba wyświetlić graczowi komunikat o wygranej, a jeśli różne – o przegranej. Do wyświetlania komunikatów służy polecenie **MessageBox.Show()**, w którym w nawiasie podajemy treść komunikatu do wyświetlenia.

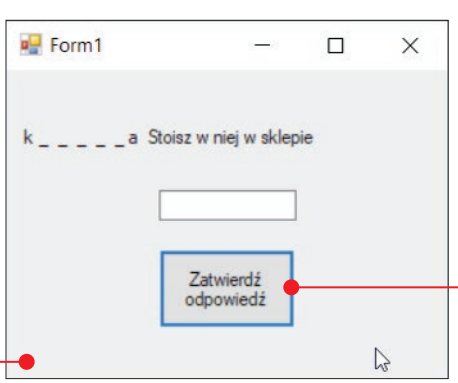
Wyrwij, użyj dziurkacza i zbieraj do segregatora

TESTOWANIE

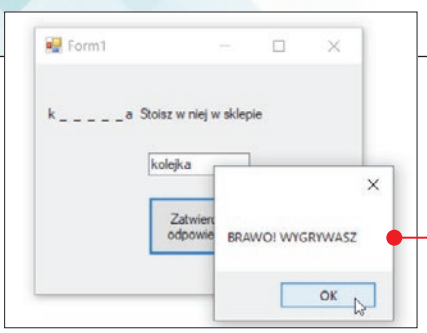
1 Przyciskiem **Rozpocznij** możemy uruchomić napisaną grę.



2 Zobaczymy wtedy nowe okno – okno gry.



3 Zależnie od udzielonej odpowiedzi po kliknięciu na przycisk **Zatwierdź odpowiedź** gra wyświetli odpowiedni komunikat.



VISUAL BASIC .NET

ZALETY PLATFORMY .NET

Z poprzedniego odcinka kursu wiemy już, że programem, w którym możemy wykorzystywać język VISUAL BASIC .NET, JEST MICROSOFT VISUAL STUDIO. TO PODSTAWA CAŁEJ PLATFORMY .NET, o której więcej dowiemy się z tego odcinka kursu, lepiej poznając język Visual Basic .NET

Platforma .NET została stworzona przez Microsoft w 2002 roku, by ułatwiać tworzenie oprogramowania na systemy Windows. Choć od jej powstania minęło już sporo lat, a technologie zmieniły się znacząco, ten cel nadal jest jednym z fundamentów **.NET Framework**.

Bardzo ważną częścią platformy jest **CLR**, czyli wspólne środowisko uruchomieniowe. To podstawa funkcjonowania platformy .NET. Dzięki niemu możliwe jest wykorzystywanie tego samego zbioru funkcji zawartych w bibliotekach przez programy napisane w różnych językach.

Ten odcinek kursu programowania został poświęcony językowi C#, który powstał na potrzeby omawianej platformy. Przedstawione dalej biblioteki i mechanizmy można wykorzystać, tworząc programy w języku C# – tak samo jak w języku Visual Basic .NET.

WŁASNA KONTROLKA WIELOKROTNEGO UŻYTKU

W poprzednim odcinku kursu, podczas projektowania wyglądu okna gry, korzystaliśmy z przybornika i znajdujących się w nim elementów, jak choćby przycisk czy pole tekstowe. Te elementy są w przyborniku umieszczone domyślnie, aby nie trzeba było za każdym razem tworzyć na nowo powtarzających się skadników programów.

A żeby uniknąć wielokrotnego tworzenia bardziej złożonych części programów, idealnie sprawdzają się własne **biblioteki DLL** – możemy je raz opracować, dodać do przybornika i wykorzystywać potem w wielu różnych projektach.

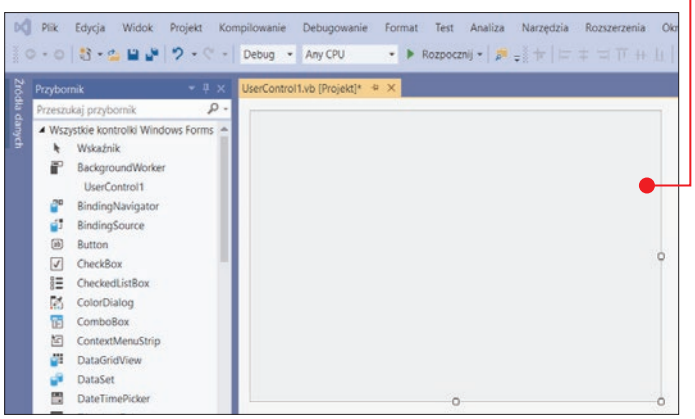
W tym poradniku, także w **Visual Studio** (DVD-KOD: **075**), krok po kroku stworzymy kontrolkę, która będzie napisana w języku Visual Basic .NET i której potem będziemy mogli używać w projektach pisanych w innych językach. Działanie naszej kontrolki będzie proste. Będzie ona pełniła rolę „stopki" okna programu. Będzie się składała z przycisku – klikając na ten przycisk, będzie można wyświetlić albo ukryć link do strony internetowej.

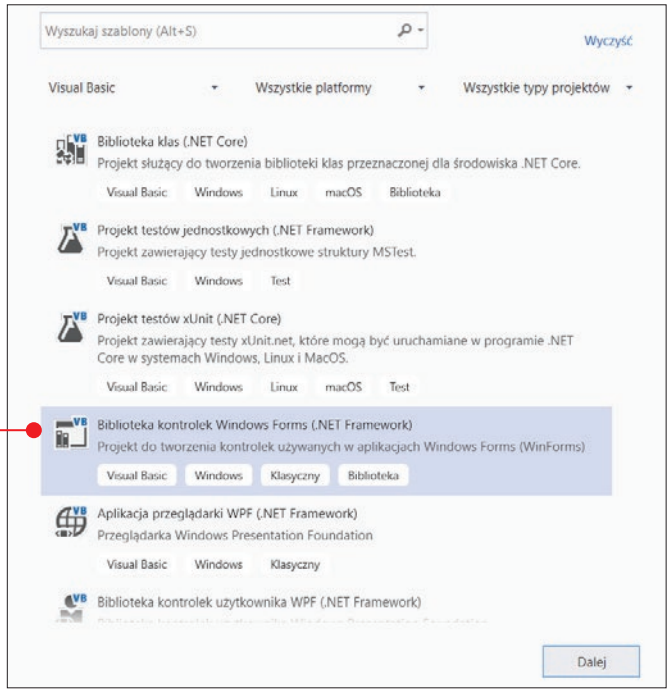
TWORZENIE NOWEGO PROJEKTU

1 Korzystając z wiedzy zdobytej w poprzednim odcinku, tworzymy nowy projekt w języku Visual Basic.

2 Podczas tworzenia projektu tym razem nie wybieramy jednak opcji **Aplikacja Windows Forms (.NET Framework)**. Zamiast tego należy wybrać typ projektu **Biblioteka kontrolek Windows Forms (.NET Framework)**.

3 Po utworzeniu nowego projektu zobaczymy, tak jak w poprzednim odcinku, obszar roboczy. Tym razem posłuży nam on do projektowania kontrolki. Podobnie, jak w wypadku tworzenia okna programu można go dowolnie powiększać i zmniejszać.

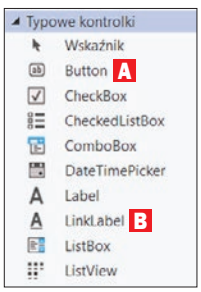




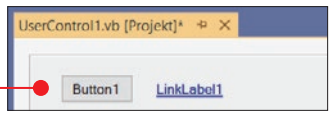
PROJEKTOWANIE KONTROLKI

By kontrolka spełniała założenia zadania, może składać się z innych, istniejących już kontrolek. Wystarczy przeciągnąć je na obszar projektowania.

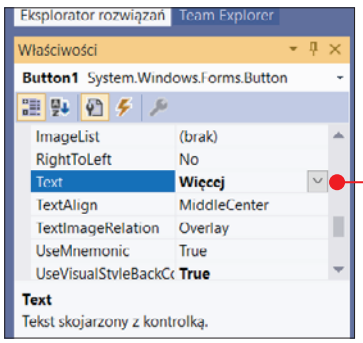
1 W tym wypadku potrzebne będą nam dwie kontrolki – **Button** A, czyli przycisk, i **LinkLabel** B, czyli etykieta stylizowana na link.



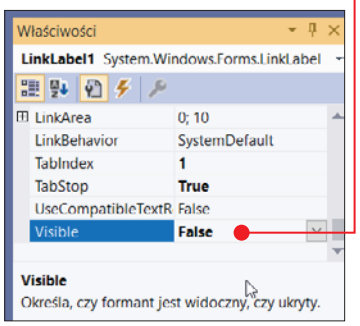
2 Przeciągamy kontrolki na obszar projektowania.

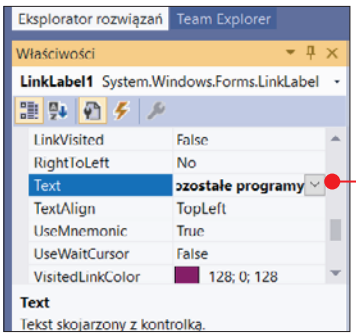


3 Poprzez sekcję **Właściwości** należy przygotować pierwotny widok tworzonej kontrolki. W tym momencie należy dokładnie przemyśleć jej działanie. Przycisk będzie odpowiedzialny za pokazywanie i ukrywanie elementu **LinkLabel**. Zatem napis na przycisku powinien zmieniać się zależnie od tego, którą z czynności wywoła kliknięcie. Może to być napis „Więcej" zamiennie z napisem „Mniej". Gdy widoczny będzie ten pierwszy, kliknięcie na przycisk powinno pokazywać **LinkLabel**, gdy zaś drugi – ukrywać. Ustawiamy właściwość **Text** elementu **Button1** na **Więcej**.



4 Skoro na przycisku jest słowo „Więcej", to etykieta z linkiem powinna pokazać się dopiero po kliknięciu na niego, a więc pierwotnie powinna być ukryta. Dla obiektu **LinkLabel1** ustawiamy właściwość **Visible** na wartość **False**.





5 Nie jest to jedyna rzecz we właściwościach tego obiektu, którą należałoby zmodyfikować. Bardzo ważny jest przekaz dla użytkownika – czyli to, co jest napisane na etykiecie – a za to odpowiada właściwość **Text**. Należałoby dać jej wartość sugerującą użytkownikowi, co wyświetli się, gdy kliknie on na link. Zatem właściwość **Text** dla obiektu **LinkLabel1** może przyjąć wartość **Odwiedź moją stronę internetową, aby pobrać kolejne programy**. Potem zaprogramujemy link tak, by kliknięcie na niego prowadziło do otwarcia określonej w skrypcie strony internetowej.

6 Po zakończeniu edycji właściwości zmniejszamy obszar projektowania tak, by nie zawierał zbędnej, pustej przestrzeni.

TWORZENIE SKRYPTU KONTROLKI

Pisanie skryptu zaczniemy od obiektu **LinkLabel1**.

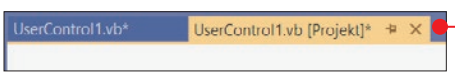
1 By móc przypisać, co ma się zdarzyć po kliknięciu na ten obiekt, klikamy na niego dwukrotnie w oknie projektowania. Spowoduje to przejście do skryptu, w którym będzie już wygenerowana metoda odpowiadająca kliknięciu na dany element.

2 Wewnątrz wygenerowanej metody można teraz zaprogramować otwieranie określonej strony internetowej. W tym celu użyjemy polecenia **Process.Start()**, w którym w nawiasie umieszcza się w cudzysłowie adres strony, jaka ma być otwarta. My dla przykładu możemy posłużyć się adresem **komputerswiat.pl**. Całe polecenie będzie wyglądało zatem następująco: **Process.Start("www.komputerswiat.pl")**.

```
Private Sub LinkLabel1_LinkClicked(sender As Object, e As L
    Process.Start("www.komputerswiat.pl")
    MessageBox.Show("Dziękuję za odwiedziny mojej strony")
End Sub
```

3 Dodatkowo można jeszcze zaprogramować wyświetlenie okna z podziękowaniem za odwiedziny strony. Tego typu komunikaty wyświetla się, korzystając z polecenia **MessageBox.Show()**, w którym w nawiasie w cudzysłowie podaje się treść komunikatu. Zatem całe polecenie może mieć postać: **MessageBox.Show("Dziękuję za odwiedziny mojej strony")**.

4 Na pasku zakładek u góry przełączamy się z powrotem do okna projektowania.



5 Klikamy dwukrotnie na przycisk, co powoduje przejście do skryptu i wygenerowanie w nim metody A, która będzie wywoływana, gdy użytkownik kliknie.

```
Public Class UserControl1
    Odwołania: 0
    Private Sub LinkLabel1_LinkClicked(sender As Object, e As LinkLabelLinkClickedEventArgs) Handles LinkLabel1
        Process.Start("www.komputerswiat.pl")
    End Sub
```

Wyrwij, użyj dziurkacza i zbieraj do segregatora

```
Public Class UserControl1
    Odwołania: 0
    Private Sub LinkLabel1_LinkClicked(sender As Object, e As LinkLabelLinkClickedEventArgs)
        Process.Start("www.komputerswiat.pl")
        MessageBox.Show("Dziękuję za odwiedziny mojej strony")
    End Sub

    Odwołania: 0
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

    End Sub
End Class
```

6 Wewnątrz tej metody należy zawrzeć instrukcję warunkową, która będzie sprawdzała: gdy będzie napis „Więcej”, kliknięcie powinno pokazać obiekt **LinkLabel1**. Piszemy zatem **If Button1.Text = "Więcej" Then**.

```
Private Sub Button1_Click(sender As Object, e As Event
    If Button1.Text = "Więcej" Then
```

7 W kolejnych liniach powinny znaleźć się instrukcje, które mają się wykonać, gdy warunek jest spełniony. Przede wszystkim powinna to być zmiana napisu na przycisku. Skoro teraz był na nim napis „Więcej”, nowym napisem powinno być słowo „Mniej”. Wpisz więc: **Button1.Text = "Mniej"**.

```
Private Sub Button1_Click(sender As
    If Button1.Text = "Więcej" Then
        Button1.Text = "Mniej"
```

```
Private Sub Button1_Click(sender As Object, e
    If Button1.Text = "Więcej" Then
        Button1.Text = "Mniej"
        LinkLabel1.Visible = True
```

8 Równie ważne jest też pokazanie etykiety z linkiem. To można zrobić, pisząc: **LinkLabel1.Visible = True**.

```
If Button1.Text = "Więcej" Then
    Button1.Text = "Mniej"
    LinkLabel1.Visible = True
Else
```

9 Instrukcja nie jest jeszcze zakończona. Dodając do niej sekcję **Else**, można określić, co ma się wykonać zaraz po kliknięciu na przycisk, gdy po kliknięciu na przycisku nie będzie napisu „Więcej” (czyli będzie tam napis „Mniej”).

10 Poleceniem **Button1.Text = "Więcej"** ponownie wstawiamy na przycisk napis „Więcej”.

```
If Button1.Text = "Więcej" Then
    Button1.Text = "Mniej"
    LinkLabel1.Visible = True
Else
    Button1.Text = "Więcej"
```

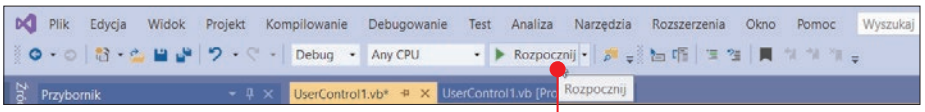
11 Link do strony internetowej, do którego daje dostęp przycisk, powinien być teraz ukryty. To osiągniemy, pisząc **LinkLabel1.Visible = False**.

```
If Button1.Text = "Więcej" Then
    Button1.Text = "Mniej"
    LinkLabel1.Visible = True
Else
    Button1.Text = "Więcej"
    LinkLabel1.Visible = False
```

12 By dopełnić całą instrukcję warunkową, zgodnie z jej składnią na końcu powinno znaleźć się jej zamknięcie – **End If**.

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    If Button1.Text = "Więcej" Then
        Button1.Text = "Mniej"
        LinkLabel1.Visible = True
    Else
        Button1.Text = "Więcej"
        LinkLabel1.Visible = False
    End If
End Sub
```

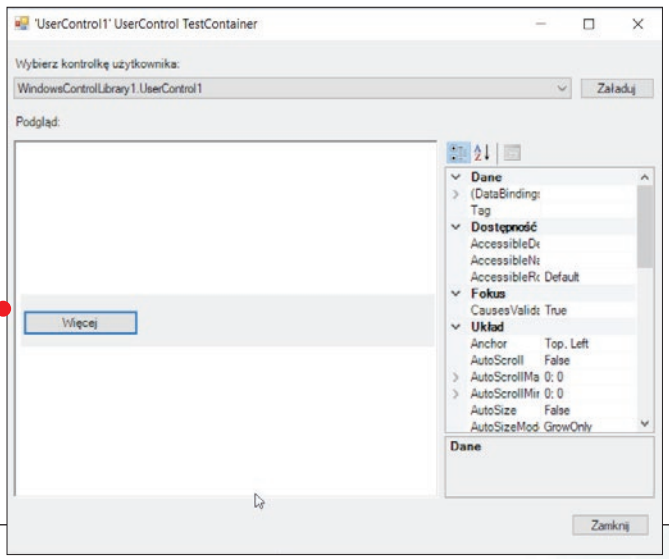
TESTOWANIE KONTROLKI



Kontrolki, podobnie jak całe programy napisane w Visual Studio, możemy na bieżąco testować.

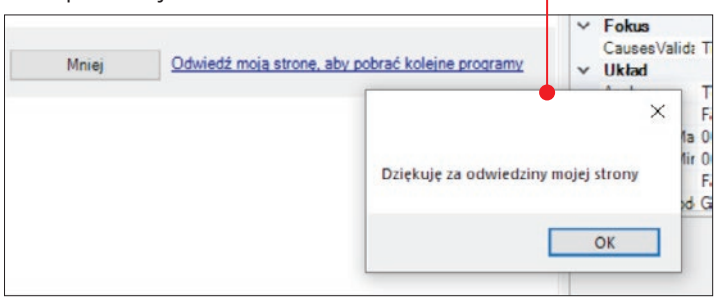
1 Po zakończeniu pisania skryptu klikamy na przycisk **Rozpocznij** znajdujący się w menu u góry okna Visual Studio.

2 Program wyświetli okno testowania kontrolek. Widać w nim nie tylko samą kontrolkę, ale też to, jakie ma ona właściwości. W sekcji **Podgląd** możemy zobaczyć, że w stanie początkowym widoczny jest jedynie przycisk.



3 Klikamy na przycisk na podglądzie – kontrolka powinna już działać i napis powinien się zmienić, a link – pokazać.

4 Również kliknięcie na link powinno spowodować otwarcie w domyślnej przeglądarce internetowej strony o podanym w skrypcie adresie. Powinno już także działać wyświetlanie komunikatu z podziękowaniami.



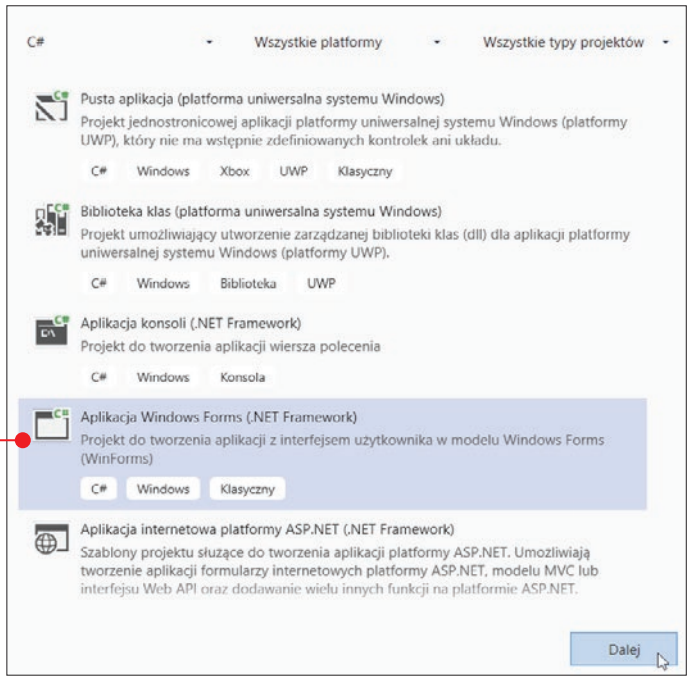
UŻYCIE KONTROLKI W INNYM PROJEKCIE

Teraz możemy też przetestować działanie kontrolki po umieszczeniu jej w innym projekcie.

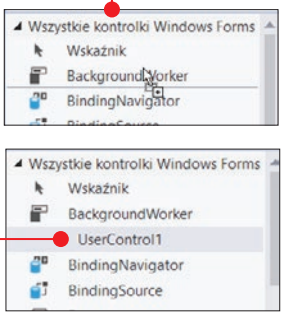
1 Tworzymy nowy projekt w Visual Studio. Tym razem powinna to być **Aplikacja Windows Forms (.Net Framework)**, jednak język projektu może być już inny. Może to być na przykład C# – mimo że kontrolka została napisana w języku Visual Basic.

2 Po utworzeniu nowego projektu znajdujemy katalog, w którym został zapisany projekt z kontrolką. Otwieramy znajdujące się w nim foldery, kolejno **bin** i **Debug**.

3 Tam znajdziemy plik z rozszerzeniem **.dll**.

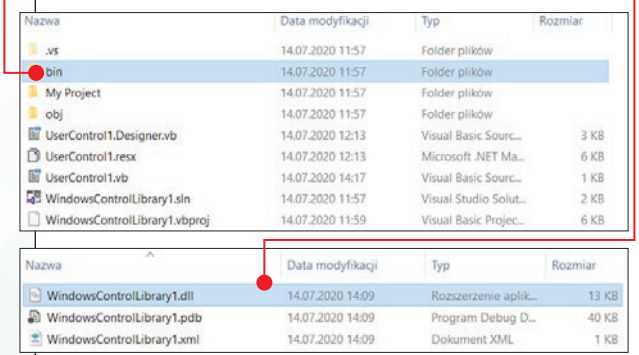


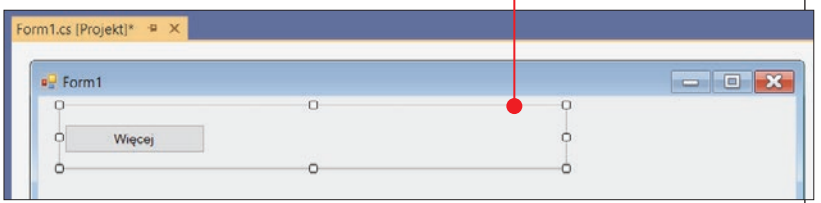
4 Przeciągamy go z folderu na **Przybornik** w nowym projekcie.



5 Kontrolka pojawi się w **Przyborniku**, skąd można ją przeciągnąć na okno tworzonego programu.

6 Po uruchomieniu projektu kontrolkę widać w oknie i można z niej korzystać.

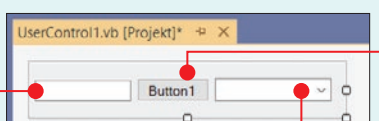


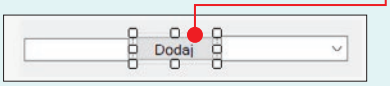


ZADANIE DODATKOWE

Stwórz własną kontrolkę, której działanie będzie polegało na dodawaniu do rozwijanej listy kolejnych pozycji wpisywanych przez użytkownika w pole tekstowe.

1 Zbudujemy wygląd kontrolki, korzystając z trzech dostępnych w Przyborniku elementów – **Textbox** (pole, do którego użytkownik wpisze pozycję do dodania), **Button** (przycisk, na który użytkownik kliknie, by zatwierdzić dodanie elementu do listy) i **Combobox** (rozwijana lista). Pamiętaj o zmianie tekstu widocznego na przycisku – tak by wskazywał on na działanie przycisku.

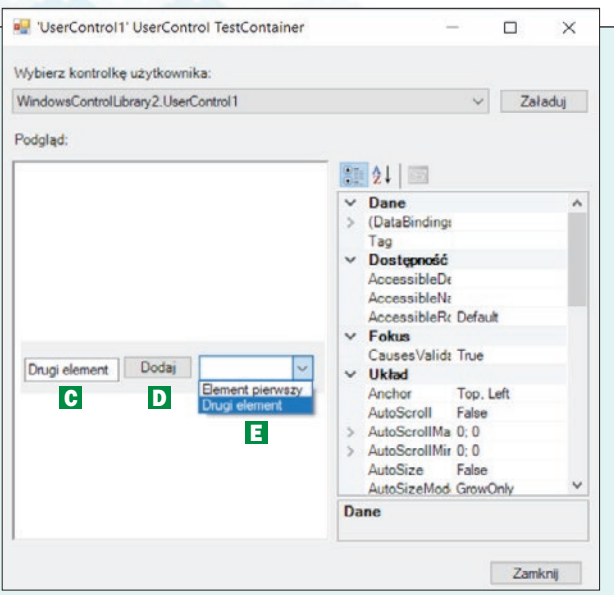




2 Generujemy metodę **A**, która wykona się po kliknięciu na przycisk.

3 W metodzie tej użyjemy polecenia, które doda do kolekcji elementów Comboboxa tekst z obiektu **Textbox1** **B**.

4 Pamiętajmy o tym, by na koniec sprawdzić działanie kontrolki. Wpisujemy tekst w pole tekstowe **C**, klikamy na przycisk **D** i rozwijamy listę **E**, by sprawdzić, czy tekst z pola na nią trafił.




```
Public Class UserControl1
    A Odwołania: 0
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

    End Sub
End Class
```

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    ComboBox1.Items.Add(TextBox1.Text) B
End Sub
```

Wyrwij, użyj dziurkacza i zbieraj do segregatora

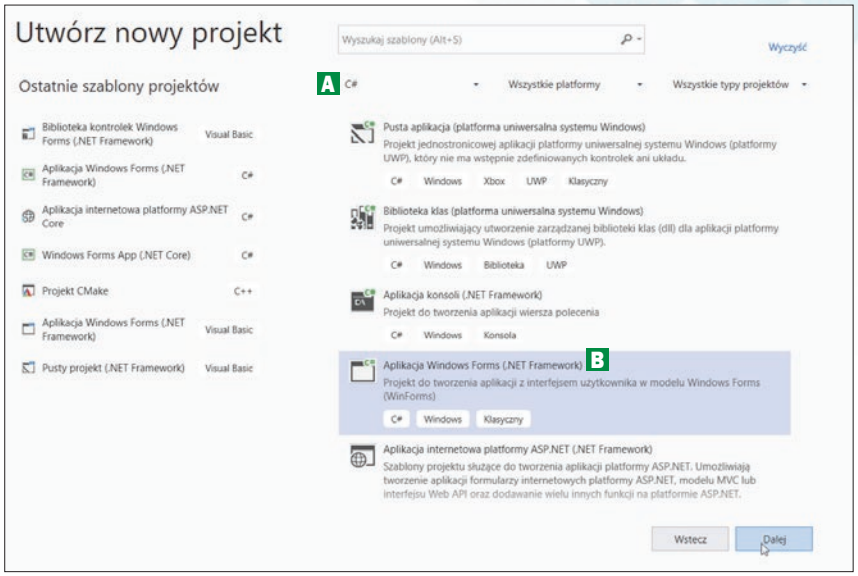
POZNAJ JĘZYK C#

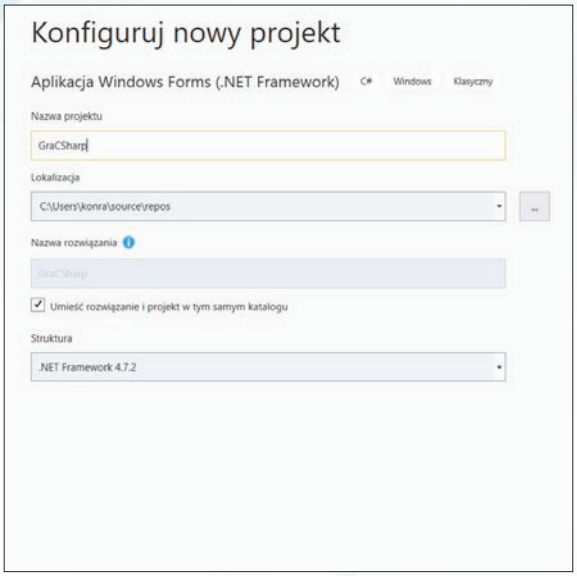
Kolejnym omawianym przez nas językiem platformy .NET jest C#. To jeden z najpopularniejszych języków programowania. Poznajmy jego składnię, tworząc prostą grę

C# to język o krótszej historii niż VB.NET (patrz 3 i 4 odcinek kursu). Jego rozwój stanowi motor napędowy dla tej platformy. Co warte podkreślenia, oba te języki to języki obiektowe, więc programowanie opiera się tu na tworzeniu klas, chociaż C# pod względem składni bardzo różni się od VB.NET. Języki te mają jednak także wiele cech wspólnych powiązanych z platformą .NET. Zaletą wynikającą z tego, że stanowią część .NET, jest także to, że w jednym i drugim języku możemy programować, korzystając z tego samego narzędzia – Visual Studio, które poznaliśmy w dwóch poprzednich odcinkach kursu.

NOWY PROJEKT W C#

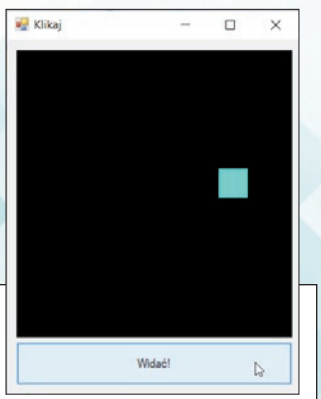
W tym odcinku, aby nauczyć się składni języka C#, stworzymy własną grę, korzystając właśnie z Visual Studio. Projekt tworzymy w podobny sposób jak poprzednio (w odcinkach 3 i 4 kursu, Niezbędnik 3 i 4/2020), ale zaczynając, musimy tym razem wybrać C# A. A ponieważ, poznając podstawy języka, pozostaniemy przy prostych aplikacjach okienkowych, dlatego wybieramy **Aplikacja Windows Forms (.NET Framework)** B.





ZASADY GRY

W grze, którą stworzymy w tym odcinku kursu, zadaniem gracza będzie klikanie na przycisk, gdy w oknie widoczny jest kolorowy kwadrat. Pojawia się on w różnych miejscach i znika cyklicznie. Za jedno kliknięcie gracz otrzyma jeden punkt. Należy jednak uważać, gdyż każde kliknięcie na przycisk, gdy kolorowy kwadrat jest niewidoczny, spowoduje wyzerowanie uzyskanej liczby punktów.



MODELOWANIE OKNA GRY

Po utworzeniu nowego projektu, tym razem dla języka C#, modelowanie okna gry odbywa się w taki sam sposób, w jaki omawialiśmy to w przypadku projektów dotyczących języka Visual Basic.NET, w poprzednich odcinkach kursu.

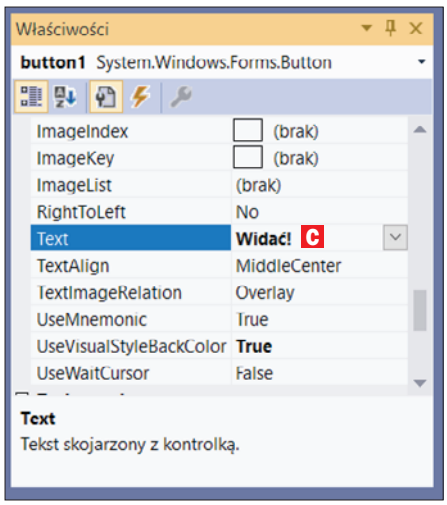
1 Mamy do dyspozycji taki sam Przybornik. To z niego należy przeciągnąć na modelowany podgląd potrzebne komponenty. Tym razem będzie to:

Button – przycisk, na który należy klikać, gdy kwadrat jest widoczny.

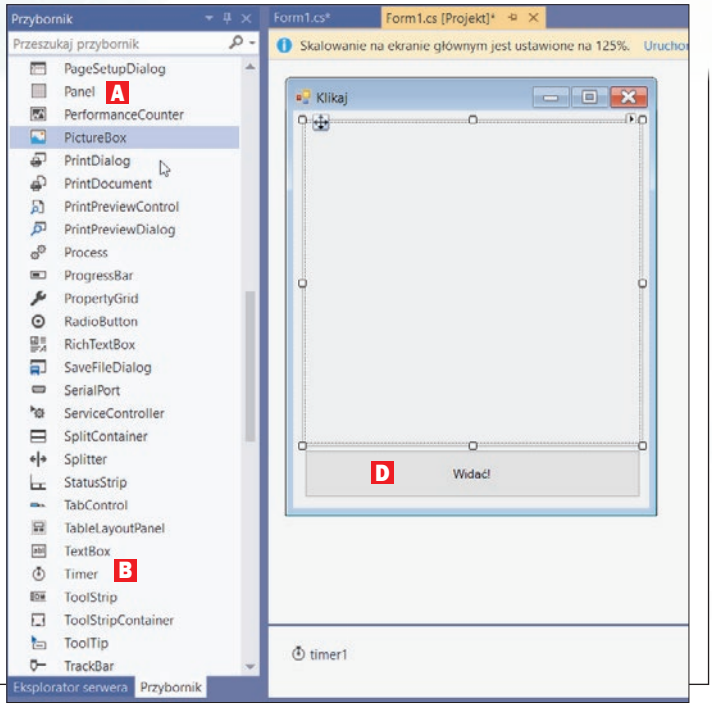
A **Panel** – płaszczyzna, na której obszarze będziemy mogli wyświetlić kwadrat.

B **Timer** – zegar, który pozwoli na odliczanie czasu do wyświetlenia i ukrycia kwadratu. Jest niewidoczny w oknie, za to daje dodatkowe zdarzenia do wykorzystania.

2 Dobieramy wielkości dla widocznych kontrolek i całego okna gry – zależnie od swoich preferencji. Pamiętajmy jednak o tym, by panel wypełniał pozostały, wolny od przycisku obszar okna.



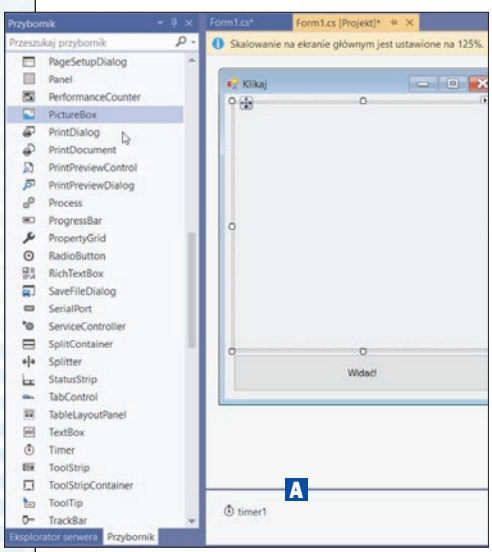
3 Warto też zmienić wartość właściwości **Text** C zarówno dla całego okna (gdy wpiszemy tam tytuł swojej gry, napis pojawi się na pasku tytułowym), jak i przycisku – umieszczając tam napis, który ma być na nim widoczny (na przykład **Widać!** D).



KODOWANIE: POJAWIAJĄCY SIĘ KWADRAT

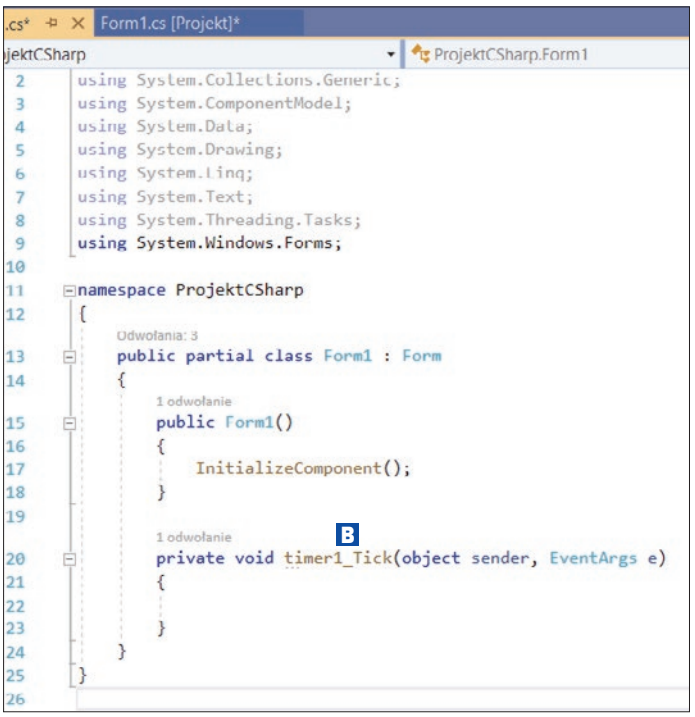
Gdy okno gry jest już zamodelowane, można przejść do tworzenia skryptów. Część z nich można wygenerować automatycznie. Zaczniemy od wygenerowania metody, która będzie wykonywała się zawsze, co pewien czas, z określonym interwałem. Do utworzenia tego mechanizmu potrzebna była kontrolka **Timer**. Teraz można z niej skorzystać.

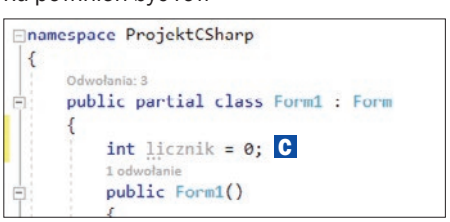
1 Kliknij dwukrotnie na kontrolkę **timer1** A pod modelowanym oknem.



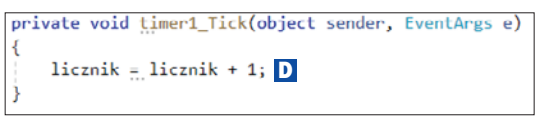
2 Nastąpi przeniesienie do kodu programu, w którym będzie już wygenerowana metoda **timer1_Tick** B. Tu warto przypomnieć sobie mechanizm **Timera**, który wykorzystywany był w naszym kursie w odcinku 2 o Small Basicu (strona 9 kursu, Niezbędnik 2/2020). W odróżnieniu od języków opartych na Basicu, w C# treść metody i innych konstrukcji wymagających wyraźnego zaznaczenia końca i początku umieszcza się w nawiasach klamrowych.

3 W klasie projektu znajduje się jeszcze konstruktor okna gry. By jakieś wartości były dostępne w całym skrypcie – w każdej jego metodzie, trzeba zadeklarować je jako pola klasy. Będzie nam potrzebne pole **licznik** – posłuży do liczenia wywołań metody **timer1_Tick**. Gdy nastąpi odpowiednia liczba wywołań, na panelu wyświetli się kwadrat. Dalej licznik będzie zerowany i wyświetlany, gdy odliczona zostanie ponownie poszukiwana liczba wywołań metody. Licznik na początku powinien być równy zero. Dlatego zapisujemy **int licznik = 0;** C. Zauważmy, że w języku C# pojedyncze instrukcje kończy się średnikiem. Inaczej niż w VB.NET wygląda też deklarowanie zmiennych i pól klasy.





4 W treści metody wywoływanej przez **Timer** należy zwiększać wartość licznika. Można to zrobić, pisząc **licznik = licznik + 1;** D.



5 Dalej w treści tej metody należałoby sprawdzać, czy licznik osiągnął już wymaganą wartość, na przykład 1000. Do tego trzeba wykorzystać instrukcję warunkową – **if**. W języku C# po słowie **if** umieszcza się nawias, w którym zapisuje się warunek, a następnie w nawiasie klamrowym – blok instrukcji, które mają się wykonać, gdy warunek będzie spełniony. Tu warunkiem jest to, że licznik będzie większy niż 1000 E.

6 Następnie należy dać programowi informację o tym, że kwadrat powinien być już widoczny. Tę informację można przecho-

Wyrwij, użyj dziurkacza i zbieraj do segregatora

```
private void timer1_Tick(object sender, EventArgs e)
{
    licznik = licznik + 1;
    if (licznik > 1000) E
    {

    }
}
```

wać w polu klasy o nazwie **widoczny**, które będzie miało typ boolowski (możliwe wartości to **true** i **false**) i początkową wartość **false** mówiącą o tym, że kwadratu jeszcze nie widać.

```
public partial class Form1 : Form
{
    int licznik = 0;
    bool widoczny = false;
```

7 Dopiero po spełnieniu warunku o odpowiednio dużej wartości licznika należy ustawić pole **widoczny** na wartość **true**.

```
if (licznik > 1000)
{
    widoczny = true;
}
```

8 Warto jeszcze zadbać o to, by kwadrat mógł znikać. Niech dzieje się to, gdy licznik przekroczy kolejną wartość. Można zapisać kolejną instrukcję warunkową, gdzie

```
private void timer1_Tick(object sender, EventArgs e)
{
    licznik = licznik + 1;
    if (licznik > 1000)
    {
        widoczny = true;
        if (licznik > 1200)
        {

        }
    }
}
```

sprawdzimy, czy licznik ma wartość większą niż 1200.

9 Z kolei, gdy ten warunek będzie spełniony, można nadać polu **widoczny** wartość **false** F, by poinformować program, że kwadrat powinien zniknąć.

```
licznik = licznik + 1;
if (licznik > 1000)
{
    widoczny = true;
    if (licznik > 1200)
    {
        widoczny = false; F
    }
}
```

10 W tym samym bloku instrukcji powinno znaleźć się też wyzerowanie licznika – **licznik = 0;** G.

```
if (licznik > 1200)
{
    widoczny = false;
    licznik = 0; G
}
```

11 Dalej przejdziemy do samego umieszczania kwadratu na panelu. Zaczniemy od wypełnienia panelu kolorem. By wypełnić go na czarno, piszemy **panel1.CreateGraphics().Clear(Color.Black);** H.

12 W kolejnej linii można zadeklarować zmienną o nazwie **bok**, która będzie służyła do przechowywania wielkości boku kwadratu – zmienna może być liczbą całkowitą (integer) i mieć wartość 30 I (bok będzie miał 30 pikseli).

13 Pozostając wciąż w tej samej metodzie, można użyć kolejnej instrukcji warunkowej, która sprawdzi wartość pola **widoczny** i przejdzie do narysowania kwadratu, gdy pole to będzie miało wartość **true** J.

```
panel1.CreateGraphics().Clear(Color.Black);

int bok = 30;

if (widoczny == true) J
{

}
```

14 Instrukcja, która odpowiada za rysowanie, wymaga podania współrzędnych (odległości od krawędzi panelu) kwadratu. Współrzędne można

```
public partial class Form1 : Form
{
    int licznik = 0;
    bool widoczny = false;
    int x = 0, y = 0; K
```

zapisać jako pola klasy – **x** i **y**, które dostaną wartość początkową równą **0** K.

15 Kwadrat można narysować poleceniem **panel1.CreateGraphics().FillRectangle(new SolidBrush(Color.Aqua), x, y, bok, bok);** L, gdzie **Color.Aqua** odnosi się do koloru wypełnienia kwadratu. Polecenie służy do wypełniania prostokątów, stąd dwukrotne użycie zmiennej **bok** – by odpowiadała zarówno za szerokość, jak i wysokość wypełnianej figury.

16 Do instrukcji warunkowej można dodać sekcję **else** M.Umieszcza się ją po zamknięciu nawiasu klamrowego z blokiem instrukcji wykonywanych, gdy warunek jest spełniony. Po słowie **else** otwiera się kolejny nawias klamrowy, gdzie umieszcza się blok instrukcji wykonywanych, gdy warunek nie jest spełniony (nawiasy klamrowe można pominąć, gdy zamiast bloku instrukcji ma się wykonać tylko jedna instrukcja).

```
private void timer1_Tick(object sender, EventArgs e)
{
    licznik = licznik + 1;
    if (licznik > 1000)
    {
        widoczny = true;
        if (licznik > 1200)
        {
            widoczny = false;
            licznik = 0;
        }
    }

    panel1.CreateGraphics().Clear(Color.Black); H
```

```
private void timer1_Tick(object sender, EventArgs e)
{
    licznik = licznik + 1;
    if (licznik > 1000)
    {
        widoczny = true;
        if (licznik > 1200)
        {
            widoczny = false;
            licznik = 0;
        }
    }

    panel1.CreateGraphics().Clear(Color.Black);

    int bok = 30; I
}
```

17 Sekcja **else** będzie odpowiadała za to, by generować nowe **x** i **y** dla odległości kwadratu od krawędzi panelu. Do generowania liczb pseudolosowych potrzebny jest obiekt klasy **Random** – utworzymy go poleceniem **Random r = new Random();** N.

```
else
{
    Random r = new Random(); N
```

18 Poprzez obiekt **r** można skorzystać z metody **Next** generującej wartość z podanego w parametrze zakresu. Powinny zostać wygenerowane odległości takie, by kwadrat całą swoją powierzchnią zmieścił się na panelu **x**, czyli odległość kwadratu od lewej krawędzi okna może być równa lub większa od zera i jednocześnie mniejsza

```
Random r = new Random();
x = r.Next(0, panel1.Width - bok); O
```

niż szerokość panelu pomniejszona o bok kwadratu. Można to zapisać instrukcją: **x = r.Next(0, panel1.Width - bok);** O.

```
panel1.CreateGraphics().Clear(Color.Black);

int bok = 30;

if (widoczny == true)
{
    panel1.CreateGraphics().FillRectangle(new SolidBrush(Color.Aqua), x, y, bok, bok); L
}
```

```
if (widoczny == true)
{
    panel1.CreateGraphics().FillRectangle(new SolidBrush(Color.Aqua), x, y, bok, bok);
}
else M
{

}
```

19 Analogicznie można zapisać to dla współrzędnej **y**, gdzie instrukcja będzie miała postać: **y = r.Next(0, panel1.Height - bok);** P.

```
if (widoczny == true)
{
    panel1.CreateGraphics().FillRectangle(r
}
else
{
    Random r = new Random();
    x = r.Next(0, panel1.Width - bok);
    y = r.Next(0, panel1.Height - bok); P
}
```

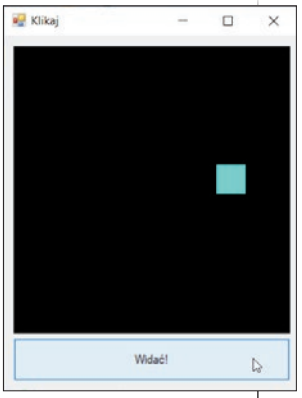
20 By Timer działał, przechodzimy do konstruktora i dodajemy w nim polecenie **timer1.Enabled = true;** R.

```
public Form1()
{
    InitializeComponent();
    timer1.Enabled = true; R
```

21 Dalej należałoby ustalić, jak często ma się wykonywać metoda wywoływana przez **Timer** – mówi o tym właściwość **timer1.Interval**, którą wyraża się w milisekundach. Przypisujemy do tej właściwości wartość 10 S – metoda wcześniej napisana będzie wykonywała się co 10 milisekund.

```
public Form1()
{
    InitializeComponent();
    timer1.Enabled = true;
    timer1.Interval = 10; S
}
```

22 Uruchamiamy aplikację – po 10 sekundach na panelu powinien pojawić się kwadrat morskiego koloru, po dwóch sekundach powinien zniknąć i pojawić się za 10 =sekund w nowym miejscu, i tak cyklicznie.



KODOWANIE: ZDOBYWANIE PUNKTÓW

Mechanizm znikających i pojawiających się kwadratów działa, ale w grze nie można jeszcze zdobywać punktów. Żeby było to możliwe, trzeba zaprogramować działanie przycisku.

1 Przechodzimy do karty modelowania okna i dwukrotnie klikamy na przycisk, by wygenerować metodę odpowiedzialną za kliknięcie na niego.

```
1 odwołanie
private void button1_Click(object sender, EventArgs e)
{

}
```

2 Wewnątrz metody trzeba dodać instrukcję warunkową, ponieważ gra będzie reagowała różnie w zależności od tego, jaką wartość ma pole **widoczny**. Gra powinna dodawać punkty, gdy **widoczny** ma wartość **true** podczas klikania na przycisk. Taki warunek należy umieścić w instrukcji warunkowej A.

```
private void button1_Click(object sender, EventArgs e)
{
    if (widoczny == true) A
    {

    }
}
```

3 By program mógł liczyć punkty, niezbędne jest zadeklarowanie w nim odpowiedniego pola do ich magazynowania. Piszemy zatem **int punkty = 0;** poza treścią metod, ale wewnątrz klasy B.

```
public partial class Form1 : Form
{
    int licznik = 0;
    bool widoczny = false;
    int x = 0, y = 0;
    int punkty = 0; B
```

4 W instrukcji warunkowej wykonywanej po kliknięciu na przycisk, gdy kwadrat jest widoczny, piszemy **punkty = punkty + 1;** C, by zwiększyć liczbę punktów.

```
private void button1_Click(object sender, EventArgs e)
{
    if (widoczny == true)
    {
        punkty = punkty + 1; C
    }
```

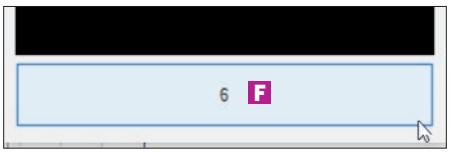
5 Do instrukcji warunkowej można dodać sekcję **else** D, w której liczba punktów będzie zerowana – gdy gracz kliknie na przycisk, kiedy nie będzie widać kwadratu.

```
private void button1_Click(object sender, EventArgs e)
{
    if (widoczny == true)
    {
        punkty = punkty + 1;
    }
    else D
    {
        punkty = 0;
    }
}
```

6 Po instrukcji warunkowej można użyć polecenia **button1.Text = punkty.ToString();** E, które wyświetli aktualną liczbę punktów na przycisku.

```
private void button1_Click(object sender, EventArgs e)
{
    if (widoczny == true)
    {
        punkty = punkty + 1;
    }
    else
    {
        punkty = 0;
    }
    button1.Text = punkty.ToString(); E
}
```

7 Przetestujmy teraz projekt – aktualna liczba punktów powinna być widoczna na przycisku F.



SKŁADNIA C#

C# różni się składnią od znanego już VB.NET. Warto przyzwyczajać się do tej składni – jest ona bardzo popularna. Na przykład instrukcja warunkowa w językach C++, C#, Java czy JavaScript będzie wyglądała tak samo.

C# jest językiem popularnym – w kolejnym odcinku kursu w następnym Niezbędniku zobaczymy, w jaki sposób wykorzystać ten właśnie język w zupełnie innym narzędziu – w silniku gier Unity. Przeczytamy, jak krok po kroku zacząć tworzyć własne gry z grafiką 3D i pisać do nich skrypty w języku C#.

VB.NET	C# / Java / C++ / JavaScript

If warunek Then
...blok instrukcji...
Else
...blok instrukcji...
EndIf
	If (warunek)
	{
	...blok instrukcji...
	}
	else
	{
	...blok instrukcji...
	}

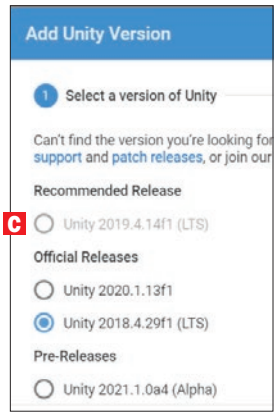
Wyrwij, użyj dziurkacza i zbieraj do segregatora

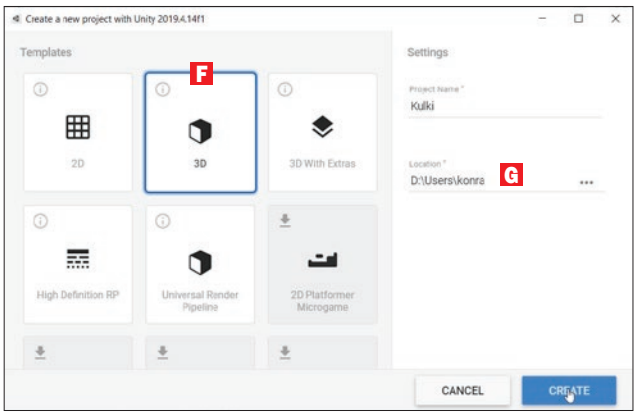
C# W UNITY

Język C# wykorzystywany jest do tworzenia skryptów w popularnym silniku gier Unity. W tym odcinku kursu programowania ponownie spotkamy się z C#, ale w innych okolicznościach – zobaczymy, jak używać go w nowym środowisku

UNITY HUB

Pracę w Unity rozpoczniemy od zapoznania się z programem **Unity Hub** (W KŚ+). To aplikacja do zarządzania projektami i wersjami silnika Unity. Trzeba mieć na uwadze to, że Unity stale ewoluuje. Zmieniają się jego możliwości i funkcje. Projekty tworzone w jednej wersji mogą nie być kompatybilne z edytorem innej wersji. Dzięki Unity Hub możemy decydować, w jakiej wersji Unity chcemy stworzyć lub otworzyć projekt.



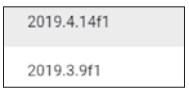


1 Wybierając z menu po lewej stronie opcję **Installs** A, możemy zobaczyć, jakie wersje są zainstalowane na naszym urządzeniu.

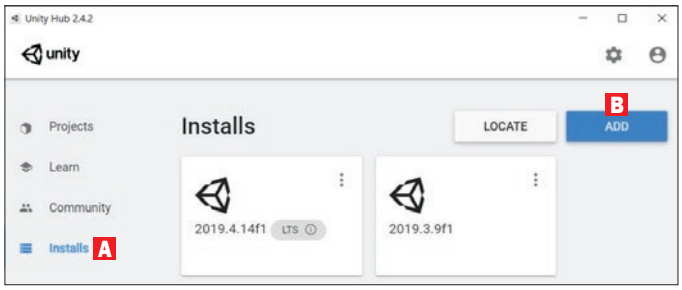
2 Klikając na przycisk **ADD** B, otworzymy okno, w którym można wybrać wersję do zainstalowania.

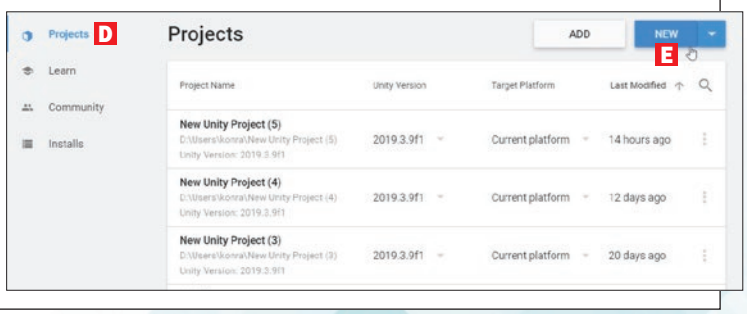
3 Aktualnie rekomendowana do używania jest wersja Unity 2019.4 – wybieramy ją C i klikamy na przycisk **Next**, by przejść do kolejnych kroków kreatora instalacji.

4 Gdy Unity w wybranej wersji jest już zainstalowane, możemy przejść do opcji **Projects** D w menu po lewej stronie. Następnie klikamy na przycisk **NEW** E, by utworzyć nowy projekt. Jeśli mamy zainstalowanych kilka wersji Unity, klikamy na strzałkę obok przycisku **NEW**, a wyświetlona zostanie lista, z której możemy wybrać, w jakiej wersji Unity chcemy stworzyć teraz nowy projekt – my wybieramy **Unity 2019.4**.



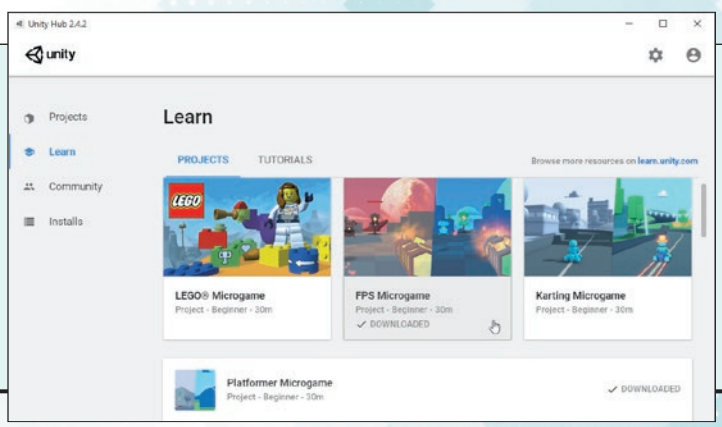
5 W nowym oknie zaznaczamy typ projektu – **3D** F, nadajemy mu nazwę G i wybieramy dla niego lokalizację. Dalej klikamy na przycisk **CREATE** i czekamy na otwarcie edytora.



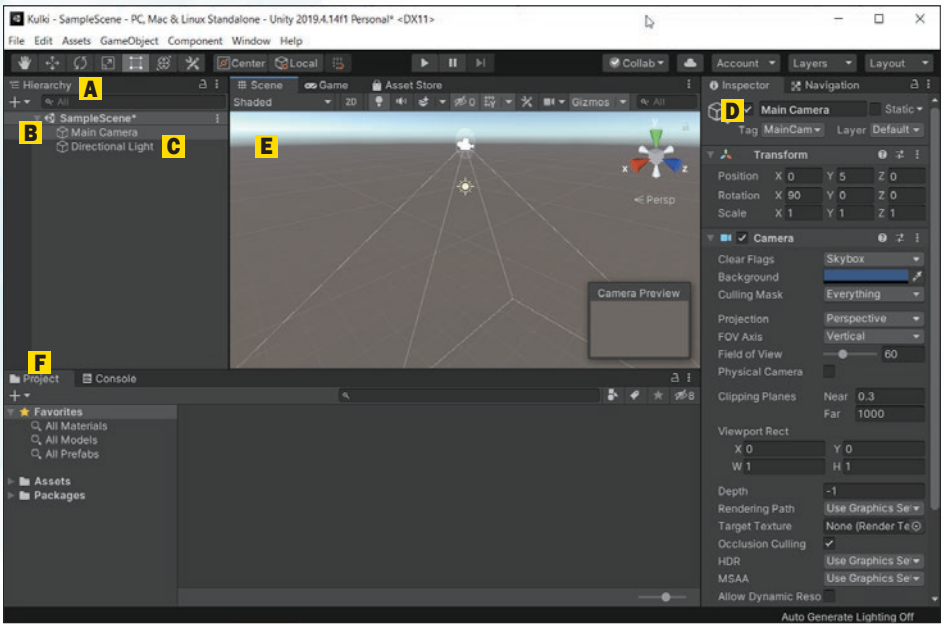


Bazowe zasoby

Gdy tworzymy projekt w **Unity Hub**, wcale nie musimy rozpoczynać pracy od pustego zasobu.
Jeśli z menu po lewej stronie wybierzemy opcję **Learn**, mamy możliwość wybrania jednego z bazowych projektów. Zawierają one wszystkie zasoby, potrzebne do tworzenia gier wybranych typów – platformowych, RPG, FPS czy wyścigów. Wśród bazowych projektów znajdziemy także takie, które pozwalają na stworzenie gry w świecie klocków Lego. W wybranych projektach znajdują się nawet gotowe do uruchomienia sceny, które można obejrzeć, by dowiedzieć się, jak zbudowane są gry, a także edytować, aby nadać im niepowtarzalny charakter.



OKNO UNITY



Po utworzeniu projektu mamy już podgląd sceny, czyli naszego głównego obszaru projektowania. Zanim jednak przejdziemy do tworzenia gry, warto dowiedzieć się, do czego służą poszczególne sekcje w oknie programu. Po lewej stronie znajduje się sekcja **Hierarchy** **A** – to hierarchia obiektów w grze. Początkowo na scenie gry są dwa obiekty:

B Main Camera (czyli główna kamera) to obiekt, który pełni rolę „oczu gracza". Tworząc naszą grę, modelujemy scenę, czyli świat gry. Niecały świat musi być widoczny dla gracza w danym momencie rozgrywki. O tym, jaki obszar i w jaki sposób jest widoczny dla gracza, decyduje właśnie kamera. Z poziomu skryptów możemy poruszać kamerą, zmieniając obraz, który jest akurat widoczny w oknie gry.

C Directional Light to światło, które służy do odpowiedniego oświetlenia obiektów sceny i jest wykorzystywane podczas renderowania gry. Bez niego wszystko byłoby niewidoczne.

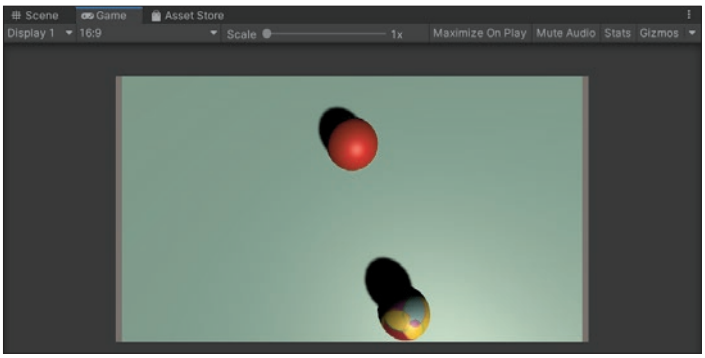
Po prawej stronie znajduje się sekcja **Inspector** **D**. Można przyrównać ją do znanej nam z MS Visual Studio sekcji Właściwości. Są tu wyświetlane właściwości zaznaczonych obiektów.

Pomiędzy hierarchią a inspektorem znajduje się **podgląd sceny** **E**. Są tutaj widoczne wszystkie dodawane obiekty. To w tej sekcji budujemy świat gry.

Nad podglądem sceny znajdują się zakładki pozwalające w jego miejscu wyświetlić podgląd gry (podgląd gry i podgląd sceny to dwie różne rzeczy), ale też przejść do **Asset Store**, czyli sklepu z zasobami do wykorzystania w tworzonych grach.

U dołu okna programu znajduje się sekcja **Project** **F**, w której znajdziemy menedżer zasobów naszego projektu. Korzystając z niego, możemy umieszczać obiekty na scenie oraz przypisywać do nich skrypty, a to jest kluczowe zagadnienie naszego kursu.

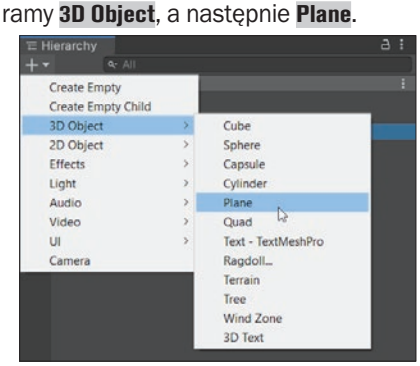
GRA – SLALOM



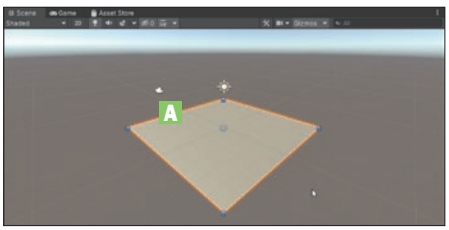
Aby lepiej poznać zarówno narzędzie, jak i język C#, stworzymy grę – slalom. Będzie ona polegała na tym, że gracz za pomocą klawiszy strzałek steruje kulką, może przesuwać ją w prawą i lewą stronę. Od góry będzie jechała na nas kulka – przeciwnik. Trzeba będzie omijać ją raz z prawej, a raz z lewej strony – na zmianę, jak w slalomie. Dla utrudnienia w każdym „przejeździe" kulka przeciwnik będzie przesuwać się w nową pozycję w poziomie.

TŁO GRY

1 Dodajemy płaszczyznę, która będzie pełniła rolę tła w grze z widokiem od góry. Klikamy na przycisk z plusem obok napisu **Hierarchy**, następnie z rozwiniętej listy kategorii obiektów możliwych do dodania wybieramy **3D Object**, a następnie **Plane**.



2 Na scenie pojawi się płaszczyzna **A**. Ma ona pełnić rolę tła naszej gry. Jeśli teraz



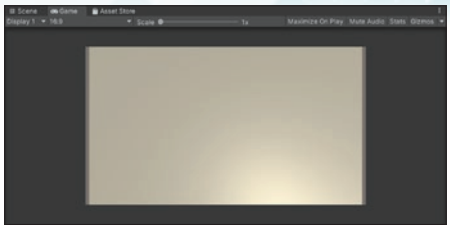
klikniemy na przycisk **Game** **B**, zobaczymy, że widok z kamery pokazuje płaszczyznę z boku. Możemy tak przestawić kamerę, by znajdowała się nad płaszczyzną i pokazywała ją z góry – w ten sposób uzyskamy taki efekt, że płaszczyzna będzie niejako tłem dla naszej rozgrywki.

3 W hierarchii zaznaczamy obiekt **Main Camera**, by w inspektorze wyświetlone zostały jego właściwości.

4 W sekcji **Transform** mówiącej o położeniu i ułożeniu obiektu należy ustawić **Position** na **X 0**, **Y 5**, **Z 0**, co sprawi, że kamera będzie znajdować się bezpośrednio nad płaszczyzną. Z kolei **Rotation** należy ustawić na **X 90**, **Y 0**, **Z 0** – dzięki temu kamera obróci się w taki sposób, że widok z niej będzie obejmował płaszczyznę.

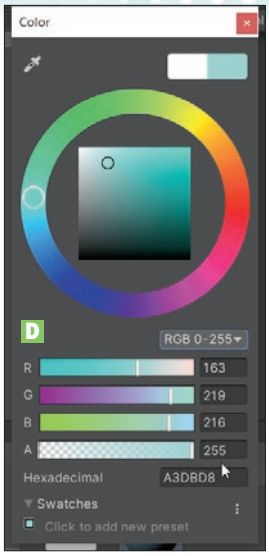
5 Po zmianach widać już na podglądzie gry, że płaszczyzna może pełnić rolę tła.

Wyrwij, użyj dziurkacza i zbieraj do segregatora

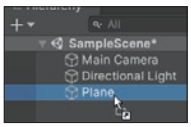


6 Płaszczyzna ma jeszcze swój domyślny kolor. Możemy go zmienić, pokrywając ją materiałem. W ten sposób nadaje się kolory i tekstury obiektom. Materiały można tworzyć. Otwieramy menu kontekstowe w sekcji **Project** i rozwijamy opcję **Create**, po czym dalej wybieramy **Material**.

7 W zasobach projektu utworzony zostanie nowy materiał, którego właściwości można zmieniać poprzez sekcję **Inspector**. By nadać materiałowi kolor, zmieniamy jego **Albedo** C, wybierając dowolny kolor D.



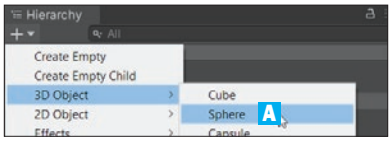
8 By nałożyć materiał na obiekt, przeciągamy materiał z sekcji **Project** na obiekt w hierarchii.



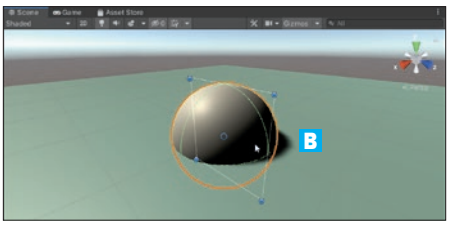
9 Płaszczyzna będzie miała już zmieniony kolor.

WSTAWIAMY KULKĘ – GRACZA

1 Dodajemy kolejny obiekt 3D do sceny, tym razem zamiast Plane wybieramy **Sphere** A. Będzie to kulka, którą gracz będzie sterował na boki.

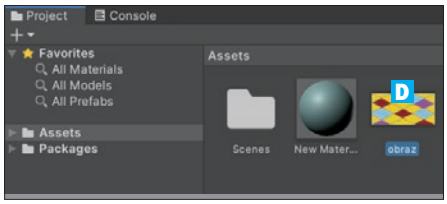


2 Kulka ustawi się na środku płaszczyzny, jednak płaszczyzna ta będzie „przecinać" kulkę B. By kulka była nad płaszczyzną w inspektorze, w sekcji **Transform** kulki zmieniamy **Y** w **Position** na **0.5**.

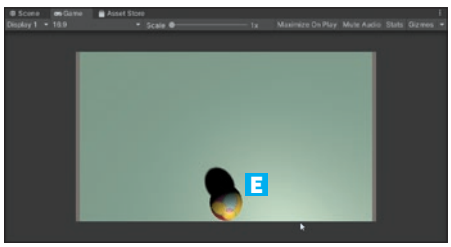


3 By kulka nie była na środku, ale u dołu widoku z kamery, zmieniamy jeszcze **Z** w **Position** na **-2** C.

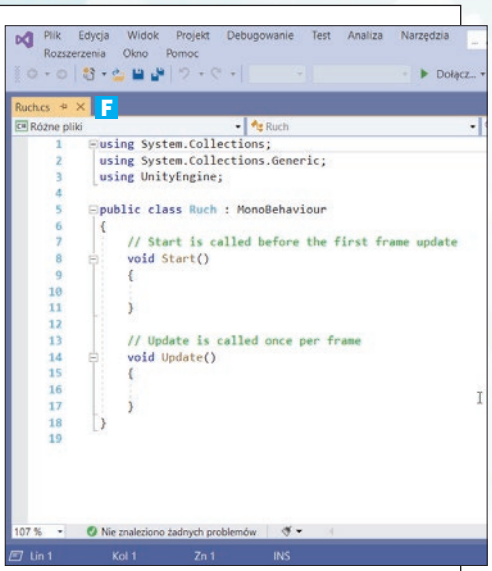
4 By uzyskać efekt toczenia się kulki, należy pokryć ją materiałem, który nie będzie jednolity. Efekt ten uzyskamy, tworząc teksturę. W folderze z projektem znajdujemy folder **Assets** i umieszczamy w nim plik graficzny – na przykład w formacie PNG. Dodany plik D pojawi się w sekcji **Project**.



5 Przeciągamy ten plik na kulkę w hierarchii – utworzony zostanie materiał z teksturą, która pokryje kulkę E.

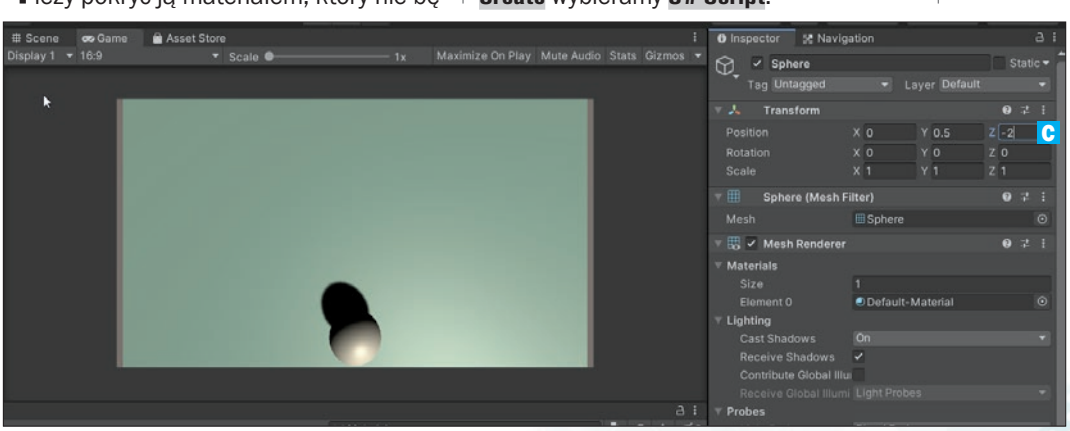


6 Do kulki trzeba teraz dołączyć skrypt, który pozwoli na sterowanie nią i sprawi wrażenie, że kulka toczy się. Otwieramy menu kontekstowe w sekcji **Project** i z pozycji **Create** wybieramy **C# Script**.



7 Nowo utworzonemu skryptowi nadajemy nazwę **Ruch** F i otwieramy go. Skrypt zostanie otwarty poprzez **Visual Studio** – to w nim edytuje się skrypty z Unity. Skrypt ten zawiera klasę **Ruch** (nazwa klasy musi być taka sama jak pliku), dziedziczącą po klasie **MonoBehaviour** – tak muszą wyglądać skrypty Unity. Dodatkowo są w tej klasie już dwie metody – **Start** i **Update**. Pierwsza wykonuje się na początku, druga – jest zapętlona i wykonuje się z każdą wyświetloną klatką gry.

```
void Update()
{
    transform.Rotate(Vector3.right); G
}
```



8 W metodzie **Update** dodajemy polecenie **transform.Rotate(Vector3.right);** G – sprawi ono, że kulka będzie rotować w kierunku symulującym jej toczenie się w stronę góry ekranu. Jeśli chcemy zwiększyć prędkość kręcenia się kulki, możemy **Vector3.right** pomnożyć przez większą liczbę, na przykład 5.

9 W tej samej metodzie powinna znaleźć się instrukcja warunkowa o następującej treści: **H**. W jej warunku zostaje sprawdzone to, czy na wejściu pojawiło się wciśnięcie klawisza – strzałki w prawo. Gdy to nastąpi, wykonywana jest zmiana pozycji, do której dodawany jest wektor w prawo. By nieco spowolnić ruch, wektor jest przemnożony przez **0.1** (litera **f** widoczna na ilustracji oznacza typ float).

10 Analogicznie, należy stworzyć instrukcję warunkową, która sprawdzi, czy wciśnięto strzałkę w stronę lewą, i w lewą stronę przesunie kulkę **I**.

H
```
if (Input.GetKey(KeyCode.RightArrow))
{
    transform.position = transform.position + Vector3.right * 0.1f;
}
```

```
void Update()
{
    transform.Rotate(Vector3.right);
    if (Input.GetKey(KeyCode.RightArrow)) I
    {
        transform.position = transform.position + Vector3.right * 0.1f;
    }
    if (Input.GetKey(KeyCode.LeftArrow))
    {
        transform.position = transform.position + Vector3.left * 0.1f;
    }
}
```

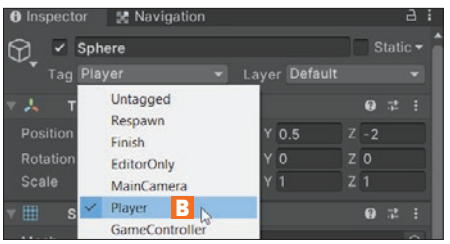
WSTAWIAMY KULKĘ – PRZECIWNIKA

1 Wiedząc, jak wstawia się obiekty 3D, dodajmy kolejny obiekt **Sphere** – będzie on pełnił rolę przeciwnika. Można zmienić nazwę obiektu, klikając na niego w sekcji **Hierarchy** prawym przyciskiem myszy i wybierając opcję **Rename**.

2 Nadajemy kulce przeciwnikowi kolor, tworząc nowy materiał, jak miało to miejsce w przypadku płaszczyzny.

3 Ustawiamy kulkę przeciwnika w pozycji startowej.

4 Tworzymy kolejny skrypt. Nazywamy go **RuchPrzeciwnika** **A**. Będzie on przypisany do przeciwnika i będzie odpowiadał za przesuwanie przeciwnika w dół okna gry i za sprawdzanie, po której stronie przeciwnika znajduje się nasza kulka. By było to możliwe, skrypt przypięty do przeciwnika musi mieć wgląd w położenie naszej kulki. Jednym ze sposobów na taki wgląd jest otagowanie kulki i wyszukanie obiektu poprzez tag z poziomu skryptu przeciwnika. Zanim otworzymy skrypt **RuchPrzeciwnika**, zaznaczamy kulkę gracza i w inspektorze, u góry w polu **Tag**, wybieramy mu tag **Player** **B**.



5 Otwieramy skrypt dla przeciwnika i edytujemy go według przykładu **C**, gdzie:
- **private GameObject g;** – tworzy w klasie pole o nazwie **g**, typem pola jest GameObject, czyli obiekt gry, a wykorzystamy je do przypisania do niego obiektu – kulki gracza.
- **private bool czyPrawa;** – tworzy pole klasy typu boolowskiego, gdzie będzie znajdować się informacja, czy gracz ma być w danym przejściu przeciwnika po jego prawej stronie.

C
```
public class RuchPrzeciwnika : MonoBehaviour
{
    private GameObject g;
    private bool czyPrawa;
    // Start is called before the first frame update
    void Start()
    {
        g = GameObject.FindGameObjectWithTag("Player");
        czyPrawa = true;
    }
```

- **g = GameObject.FindGameObjectWithTag("Player");** – przypisuje do obiektu o nazwie **g** obiekt ze sceny, który ma ustawiony tag **Player**.
- **czyPrawa = true;** – nadaje polu **czyPrawa** wartość początkową **true**, co oznacza, że nasza kulka będzie musiała być po prawej stronie przeciwnika w pierwszym przejściu.

6 Metoda **Update** w klasie **RuchPrzeciwnika** powinna mieć postać **D**, gdzie:
- **transform.position = transform.position + Vector3.back * Time.deltaTime;** – przesuwa przeciwnika w dół okna gry. Dodatkowo, wektor przesunięcia przemnożony jest przez **Time.deltaTime** – to specjalna wartość, przez którą, zgodnie z dobrą praktyką, należałoby mnożyć cykliczne przesunięcia obiektów wywoływane w funkcjach **Update** różnych skryptów. Update wykonuje się z wyświetleniem każdej klatki gry. Na mocnym sprzęcie wykonuje się zatem częściej, a na słabszym rzadziej. To może doprowadzić do sytuacji, że różna będzie prędkość gier na różnych sprzętach. By uniknąć takiej sytuacji, krok przesunięcia uzależnia się od częstotliwości wykonywania skryptu.
- **if (transform.position.z <=-3)** – sprawdzamy, czy przeciwnik jest już w takiej pozycji, że zjechał ze sceny jej dolną krawędzią.
- **if (czyPrawa == true)** – sprawdzamy, czy mamy być po prawej stronie przeciwnika.
- **if (g.transform.position.x < transform.position.x)** – sprawdzamy, czy zamiast być po stronie prawej, nie znaleźliśmy się po stronie lewej. Gdy ten warunek jest spełniony, oznacza to, że pomyliliśmy strony.
- **Destroy(g);** – myląc stronę, usuwamy kulkę gracza,...
- **Destroy(this);** – ...a także kulkę przeciwnika.
- **czyPrawa = false;** – zmieniamy stronę, gdzie ma być kulka w kolejnym przejściu, na lewą.
- W sekcji **else** opisujemy, co ma się stać, jeśli kulka ma być po lewej stronie przeciwnika. Odwracamy warunek w instrukcji warunkowej, sprawdzając, czy zamiast być po lewej, nasza kulka nie znalazła się po prawej. Jeśli tak jest, obydwie kulki są usuwane.
- **czyPrawa = true;** – ustalamy, że w kolejnym przejściu nasza kulka ma być po prawej stronie przeciwnika.
- **transform.position = new Vector3(Random.value * 6 - 3, transform.position.y, 3);** – teleportujemy kulkę przeciwnika na górę okna gry, nadając jej nową pozycję. Na osi **X** pozycja jest losowana: **Random.value** zwraca ułamek z przedziału od 0.0 do 1.0 – jeśli pomnożymy tę wartość razy 6 i od wyniku odejmiemy 3, zakres wygenerowanej liczby zmienia się na: od -3.0 do 3.0.

Ważne: by uzyskać gotowy plik z grą, klikamy na **File**, wybieramy **Build And Run**; po wybraniu folderu do zapisania gry zostanie utworzony i uruchomiony program – gra.

D
```
void Update()
{
    transform.position = transform.position + Vector3.back * Time.deltaTime;
    if (transform.position.z <=-3)
    {
        if (czyPrawa == true)
        {
            if (g.transform.position.x < transform.position.x)
            {
                Destroy(g);
                Destroy(this);
            }
            czyPrawa = false;
        }
        else
        {
            if (g.transform.position.x > transform.position.x)
            {
                Destroy(g);
                Destroy(this);
            }
            czyPrawa = true;
        }
        transform.position = new Vector3(Random.value * 6 - 3, transform.position.y, 3);
    }
}
```

Wyrwij, użyj dziurkacza i zbieraj do segregatora